НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютеризовані системи управління»
спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»
на тему: «Мобільний застосунок пошуку та покупки розважальних сервісів міста»

Виконав:
студент 4 курсу, групи ІА-62
Рязанцев Андрій Андрійович _____

Керівник роботи:
к.т.н., доцент
Креденцар Світлана Максимівна _____

Рецензент:
к.т.н., доцент
Лобанчікова Надія Миколаївна _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент_____
(підпис)

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 151 «Автоматизація та комп'ютерно-інтегровані технології»

Освітньо-професійна програма «Комп'ютеризовані системи управління»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Рязанцеву Андрію Андрійовичу

1. Тема проєкту «Мобільний застосунок пошуку та покупки розважальних сервісів міста», керівник проєкту Креденцар Світлана Максимівна, к.т.н, доцент, затверджені наказом по університету від №1081-с від 7 травня 2020р.

2. Термін подання студентом проєкту 09.06.2020

3. Вихідні дані до проєкту мобільний застосунок пошуку та покупки розважальних сервісів міста, який надає можливість пошуку та відображення подій на мапі, а також підключені модулі платіжної системи та соціальної мережі.

4. Зміст пояснювальної записки вступ, аналіз існуючих рішень та формування задач розробки, системи та принципи розробки мобільного додатку, моделювання та конструкція мобільного додатку, розробка веб-інтерфейсу додатку.

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо) діаграма архітектури додатку, діаграма бази даних, діаграма класів, діаграма навігації

6. Дата видачі завдання 07.03.2020

Календарний план

№ з/п	Назва етапів виконання дипломної проєкту	Термін виконання етапів проєкту	Примітка
1	Аналіз існуючих рішень та формування задач розробки	07.03 - 14.03	
2	Системи та принципи розробки мобільного додатку	15.03 - 28.03	
3	Моделювання та конструкція мобільного додатку	29.03 - 12.04	
4	Розробка серверної частини додатку	13.04 - 19.04	
5	Розробка інтерфейсу додатку	20.04 - 03.05	
6	Підключення платіжної та соціальної систем	04.05 - 17.05	
7	Оформлення тектової документації	18.05 - 24.05	

Студент Андрій РЯЗАНЦЕВ

Керівник Світлана КРЕДЕНЦАР

АННОТАЦІЯ

Рязанцев А.А. Мобільний застосунок пошуку та покупки розважальних сервісів міст

Пояснювальна записка дипломного проекту складається з п’яти розділів, містить 84 сторінок, 45 рисунків, 6 таблиць, 4 додатки, 15 джерел та 4 кресленика.

Ключові слова: подія, мапа, http запити, Find&Go, компонент, інтеграція.

Дипломний проект присвячений розробці мобільного додатку пошуку події.

Метою розробки додатку є спрощення процесу пошуку бажаних подій та придбання квитків для користувача.

У розділі аналізу існуючих рішень та формування задач розробки описано середу розробки додатку та виявлені переваги середи та вибір мови розробки. Проведений аналіз існуючих додатків IT-проектів та виявлені основні недоліки. Опис предметної області, та постанова задач розробки.

У розділі системи та принципи мобільного додатку були створені структури модулів та їх загальна взаємодія. Створення архітектури інтеграції соціальної мережі та платіжної системи. Проведений аналіз та використання зовнішніх бібліотек. Була розроблена структура бази даних, яка відповідає поставленим задачам проекту.

У розділі моделювання та конструкція мобільного додатку була представлена реалізація архітектури серверу, підключення бази даних та налагодження взаємодії вхідних даних, створення архітектури маршрутизації додатку та Api запитів.

У розділі розробка веб-інтерфейсу додатку був розроблений інтерфейс з інтеграцією соціальної мережі та платіжної системи, який відповідає усім постановам розробляємого проекту.

У розділі інструкція користувача був проведений детальний опис кожного шагу користувача при використанні мобільного додатку.

ABSTRACT

Ryazantsev A.A Mobile application for finding and buying entertainment services in cities

Structure and scope of work. The explanatory note of the diploma project consists of five sections, contains 84 pages, 45 figures, 6 tables, 4 appendices, 15 sources.

Project keywords: event, map, http requests, Find&Go, components, integration.

Thesis project is dedicated to the development of a mobile event search application. The purpose of developing the application is to simplify the process of finding desired events and purchasing tickets for the user.

The section of analysis of existing solutions and formation of development tasks describes the environment of application development and identifies the advantages of the environment and the choice of development language. The analysis of existing applications of IT projects is carried out and the main shortcomings are revealed. Description of the subject area, and the solution of development tasks.

In the section of the system and the principles of the mobile application, the structures of the modules and their general interaction were created. Creating an architecture for the integration of social networks and payment systems. The analysis and use of external libraries is carried out. A database structure was developed that meets the objectives of the project.

The section of modeling and design of the mobile application presented the implementation of the server architecture, database connection and input interaction, creating the architecture of application routing and Api queries.

In the section of web application development the interface was developed with the integration of social network and payment system, which corresponds to all the provisions of the developed project. The user manual section provided a detailed description of each user step when using the mobile application.

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІА62.180БАК.005 ПЗ	Мобільний застосунок пошуку та	84		
6			покупки розважальних сервісів міста			
7			Пояснювальна записка			
8						
9	А3	ІА62.240БАК.005 Д1	Мобільний застосунок пошуку та	1		
10			покупки розважальних сервісів міста			
11			Діаграма архітектури додатку			
12						
13	А3	ІА62.240БАК.005 Д2	Мобільний застосунок пошуку та	1		
14			покупки розважальних сервісів міста			
15			Діаграма навігації користувача			
16						
17	А3	ІА62.240БАК.005 Д3	Мобільний застосунок пошуку та	1		
18			покупки розважальних сервісів міста			
19			Діаграма класів та моделей			
20						
21	А3	ІА62.240БАК.005 Д4	Мобільний застосунок пошуку та	1		
22			покупки розважальних сервісів міста			
23			Діаграма бази даних			
24						
25						
26						
27						
28						

Зм.	Аркуш	№ докум.	Підпис	Дата	ІА62.240БАК.005 ТП		
Розроб.		Рязанцев			Мобільний застосунок пошуку та покупки розважальних сервісів міста Відомість технічного проекту	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Креденцар				Т	
Реценз.						КПІ ім. І. Сікорського група ІА-62	
Н. Контр.							
Затв.							

Пояснювальна записка
до дипломного проєкту
на тему: «Мобільний застосунок пошуку та придбання розважальних сервісів міста»

Київ – 2020 року

ЗМІСТ

ВСТУП

У наш час неможливо уявити наше цивілізоване життя без мобільних додатків. Мобільні додатки невід'ємна частина людства. Це комунікації між людьми, подорожі, відвідування різноманітних закладів, спостерігання за соц. мережами, та багато інших функцій. Людям більше не потрібно чекати поки до адресата дійде його паперовий лист, смс-повідомлення або повідомлення в месенджері доставляються миттєво.

В 20 сторіччі нам більше не потрібні дошки оголошень чи реклама по телевізору різноманітних заходів. Все це може бути в смартфоні будь-якого користувача. За останні 10 років кількість різноманітних заходів у світі збільшилася на 84%, так у 2010-му році понад 247 000 тисяч зареєстрованих фестивалів, у 2020-му році цифра добігла 1 729 000 тисяч зареєстрованих фестивалів.

Під час обрання теми розробки додатку, я поставив собі запитання чи буде мії додаток надавати якусь важливу користь для того щоб користувач їм корисутвався. Існує багато додатків які надають можливість користувачу читать про поточні події або сервіси для прибання квітків на цікавлячу подію але в Украї немає ніяких аналогів пошукової системи з відображенням події на мапі та придбанням квитків.

Соціальне опитування надає інформацію про те що користувачі мобільних додатків позитивно реагують на наявність мапи у самому додатку. Додавання мапи є невід’ємною частиною розробки, можливість бачити події які знаходяться поблизу місцезнадження корисутвача.

Розглянути дану систему можна не лише для придбання квитків, але і як соціальну мережу. Нинче суспільство не може уявити своє життя без смартфону, а соціальні мережі є їх невід’ємною частиною. Мова йде про додавання частини соціальнох мережі щоб користувачі мали можливість спілкуватися та ділитися бажаними подіями.

Мета роботи полягає в розробці додатка з функцією відстеження геолокації різноманітних закладів за місцем знаходження користувача на карті на базі розробки IOS. Розроблений проект допоможе людям у пошуку заходів, поряд з ними, за для комфортного проведення часу.

Розглянута система повинна забезпечувати роботу функціоналу додатку, а саме:

- створення та проектування веб-серверу
- створення та налаштування інтерфейсу додатку
- створення бази даних
- підключення соціальної мережі та платіжної системи

Розроблений додаток має надавати весь функціонал додатку кожному користувача, а також бути реалізований в простому та зрозумілому користувачу форматі.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ФОРМУВАННЯ ЗАДАЧ РОЗРОБКИ

Метою розробки додатку є збереження часу та ресурсів, які користувачі постійно витрачають на моніторинг сервісів для покупки квитків чи онлайн-афіш, блогів та іншого. Сучасні великі міста постійно ростуть і змінюються. Відслідковувати нові течії мистецтва стає досить важко. Тим паче, якщо користувач - турист, який шукає розваг на вихідні.

Розроблений додаток зачіпає розважальну частину життя людини. А головне питання, яке він вирішує - це допомога людям у спрощеному пошуку бажаних заходів. Тому ідеальним рішенням буде “розумна бібліотека”, яка в собі містить всі дані про поточні та майбутні події і може їх сортувати та пропонувати: відштовхуючись від смаків користувача. Слід виокремити головні критерії для написання додатку - доступність у використанні та простота пошуку.

З точки зору користувача, важливим аспектом є впевненість в корисності та логічності модулів та функцій розробленого додатку. Якщо програма не несе за собою правильного інформаційного навантаження - то вона не має ніякої цінності для користувача, і у нього не буде необхідності його використовувати. Якщо додаток корисний, проте вимагає багато часу і зусиль, люди просто не стануть витрачати свій час та намагатися його вивчити [1]. З допомогою розробленого додатку темер можна отримувати інформацію про заходи в режимі реального часу, на всій продукції створені на платформі IOS.

За допомогою простого і зручного інтерфейсу - користувач може фільтрувати найближчі до його геопозиції події. А також додавати будь-який захід в список бажаних, налаштовувати повідомлення про їх наближення. У додатку, для користувача продумана можливість соціальної активності [2]. Кожен може опублікувати в соц. мережу Facebook свої плани

на відвідування тих чи інших місць, а після, оцінити та поділитися враженнями.

1.1 Аналіз існуючих IT проектів

Метою аналізу є підкреслення недоліків існуючих проектів для формування плану розробки додатку. До розгляду бил взяті найсмачніші та найпопулярніші рішення сьогодення: Resident Advisor та Facebook Events.

1.1.1 Мультиплатформа Resident Advisor

Мультиплатформа Resident Advisor - це онлайн музичний журнал та громадська платформа, яка створена для демонстрації подій, пов’язаних з електронною музикою та її виконавцями, у всьому світі. Платформа Resident Advisor збражена на рисунку 1.1.

Рисунок 1.1 — Інтерфейс додатку Resident Advisor

Мобільний додаток також управляє сервісами, які включають в себе списки подій, продаж квитків, довідники клубів та промоутерів, фотогалереї, профілі виконавців та звукозапису, діаграми DJ, онлайн-спільноту.

Недоліки Resident Advisor:

— унікальна платіжна система не піддається інтегруванню популярних платіжним методів;

— складний інтерфейс;

— відсутність графічного інтерфейсу, мапи;

— відсутність локалізації користувача.

1.1.2 Додаток Facebook Events

Додаток у соціальній мережі Facebook має цілий розділ організації заходів, який зображено на рисунку 1.2. Він надає можливість користувачам повідомляти друзям про майбутні події у своїй громаді та організувати соціальні збори. Для публікації події потрібна назва події, мережа, ім'я хоста, тип події, час початку, місцеположення та список запрошених друзів.

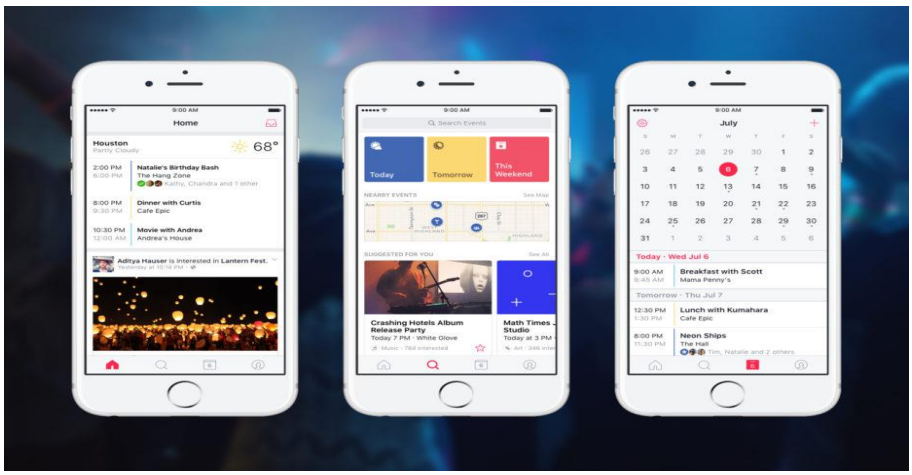

Рисунок 1.2 — Інтерфейс додатку Facebook Events

Недоліки Facebook Events:

— приватність події;

— відсутність мапи;

— функціонал реалізований як частина соціальної мережі;

— складне створення бажаємих подій;

— відсутність придбання квитків для бажаної події.

1.1.3 Порівняльна характеристика

Таблиця 1.1 – Порівняльна характеристика додатків

Крітерії	Додатки		
	Resident Advisor	Facebook Events	F&G
Залежність подій від геолокації	-	-	+
Придбання квитків	+	-	+
Зручність інтерфейсу	-	-	+
Приватність заходів	-	+	+
Частина соціальної мережі	-	+	+

На основі приведеної вище таблиці, можна зробити висновок, що розроблений додаток має загальний функціонал, відповідний до існуючих додатків та були враховані та виправлені усі недоліки. начною його перевагою над аналогами є поєднання двох основних параметрів в одному місці, а саме, залежність від геолокації та зручний інтерфейс [3].

1.2 Змістовий опис і аналіз предметної області

Одне з перших питань на початку розробки додатку це середовище. Середовищ для розробки мобільного додатку є дуже багато, але обраною програмою була Xcode через свої сущесні переваги. Xcode включає всі інструменти, необхідні для створення програми в одному програмному пакеті а саме:

— текстовий редактор;

— компілятор та система збірки. За допомогою Xcode можливо писати;

— компілювати та налагоджувати додаток, надіслати його в магазин додатків Apple;

— він містить ряд інструментів, що допомагають швидко розвиватися, тому досвідчені розробники можуть створювати програми блискавично, а початківці стикаються з меншою плутаниною та перешкодами для створення чудового додатку.

Головним питання для початку розробки додатку було обрання платформи. Були розглянуті 2 операційні системи для смартфонів IOS та Android.

Переваги та недоліки платформ IOS і Android представлено в таблиці 1.2.

Таблиця 1.2 – Порівняння платформ IOS та Android

Критерії	Платформи	
	IOS	Android
Якість графічного дизайну	+	-
Налаштування девайсу	-	+

Критерії	Платформи	
	IOS	Android
Регулярність оновлення ПО	+	-
Надійність та захист даних	+	-
Доступність	-	+
Екосистема продукції	+	-
Фасування даних між продукцією	+	-

Виходячи з аналізу таблиці платформа IOS має значні переваги у функціональній галузі. З часом кожна з платформ поступово походить одна на одну, Android тим що додає більше функціонал та зручності до своїх додатків, а IOS стає більш доступним до користувачів, також працює над функціональністю та можливістю індивідуального налагодження пристрою. За основу розробки додатку була обрана ОС IOS [7].

Наступне важливе питання це обрати мову розробки. Як редактор коду, Xcode підтримує величезну кількість мов програмування - C, C ++, Objective-C, Objective-C ++, Java, AppleScript, Python, Ruby, ResEdit та Swift, JavaScript.

Усім відома мова розробки JavaScript, популярна та дуже многофункціональна. Можливості цієї мови вже переросли усі очікування, розробку мобільних додатків це теж доторкнулися.

JavaScript це мова з великими перевагами, які роблять його найкращим вибором серед подібних йому, особливо в деяких варіантах застосування. Всього кілька переваг використання JavaScript:

— компілятор додатку інтерпретований до веб-браузеру та HTML сторінки;

— простий у використанні та вивченні;

— має обширний функціонал відладки додатків;

— велика бібліотека відстежування та налаштування події до HTML сторінки;

— JS – мультиплатформна мова, має можливості використовувати у будь-яких браузерах;

— можливість використовувати JavaScript для перевірки на правильне формотування вхідних даних і зниження необхідності ручної перевірки даних;

— надає функціональних можливостей додатку;

— простота інтегрування;

— однопоточна мова що спрямована на швидкість роботи.

1.3 Формування та постановка задач.

На основі розглянутих IT-проектів були виявлені та враховані усі недоліки існуючих додатків. Завдяки аналізу предметної області можливо постановити основні задачі розробки додатку:

— створити мобільний додаток, який надає інформацію про події користувачу.

— створити та налаштувати систему безпеки додатку, задля запобігання неавторизованого доступу до системи.

— розробити та підключити веб-сервер для можливості взаємодії з базою даних та забезпечити роботу з Api.

— створити та підключити базу даних до серверу, з метою зберігання даних.

— розробити інтерфейс додатку, для зручного користування системою користувачем.

— інтегрувати карту до інтерфейсу, для відображення маркерів подій.

— підключити та налаштувати платіжну систему у додаток, задля реалізації функціоналу придбання квитків.

Висновки до розділу

Суть цього проекту - створити додаток, який стане універсальною, новою платформою спілкування та пошуку подій за вподоби користувачу, задля спрощення життя користувача, це і є основною метою мобільних додатків. У розділі проведений аналіз існуючих IT-проектів з подібним функціоналом, розглянуті основні функції та виявленні основні недоліки, які були враховані для розробки унікального додатку.

Мобільний додаток повинен мати налаштовану локалізацію та графічну тему. Сервер додатку повинен бути в змозі налаштувати типи запитів для авторизації та аутентифікації користувача.

2 СИСТЕМИ ТА ПРИНЦИПИ РОЗРОБКИ МОБІЛЬНОГО ДОДАТКУ

Зараз розробка мобільних додатків - одне з найпопулярніших завдань у галузі інформаційних технологій. Мобільна розробка має на меті створити додатки, які можуть принести користь споживачам. Вирішить завдання, алгоритми для вирішення якого було неможливо заздалегідь. Тепер програми можуть виконувати аналіз інформації з різних джерел, допомагати користувачам приймати рішення, контролювати процеси та виконувати інші важливі завдання самостійно за найменший час та аналіз. Це, у свою чергу, допомагає впорядкувати бізнес-процеси та підвищити продуктивність та ефективність прийняття рішень.

Структура взаємодії архітектури розробляємого додатку представлена на рисунку 2.1

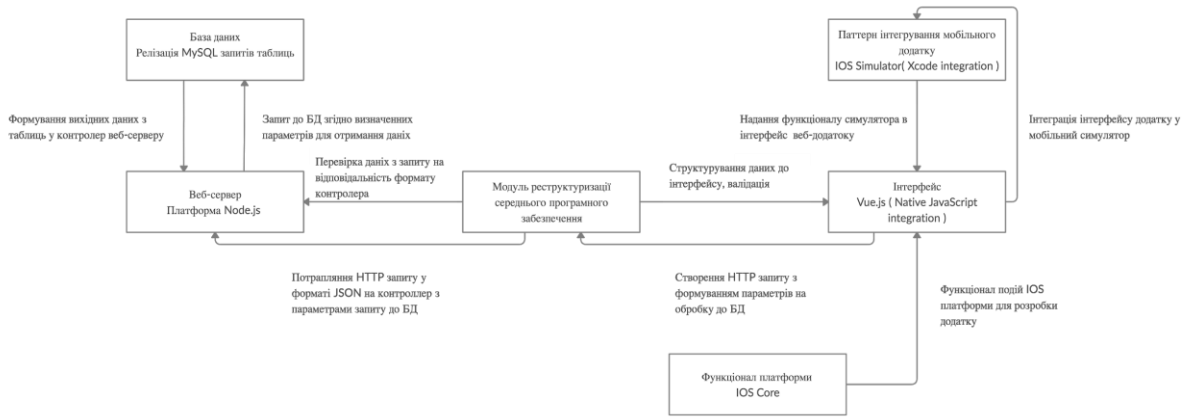

Рисунок 2.1 — Архітектура розробляємого додатку

Рівні роботи додатку представленні на рисунку 2.1 та поділяються на:

— фунціонал платформи IOS Core

— взаємодія веб-серверу та інтерфейсу

— робота бази даних та використання інструменту середовища розробки IOS Simulator

2.1 Моделювання та аналіз програмного забезпечення

За основу розробки додатку була взята платформа IOS - ця платформа має багатофункціональне середовище для створення модульного додатку. Архітектура роботи IOS додатку зображена на рисунку 2.2

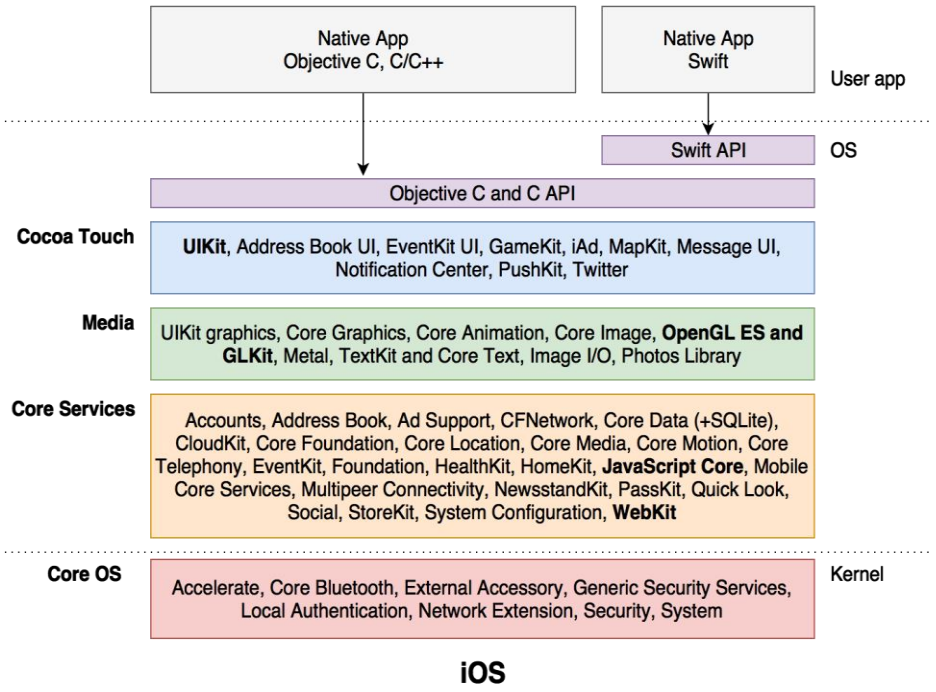

Рисунок 2.2 — Архітектура програмного забезпечення IOS

2.1.1 Моделювання IOS Core

IOS Core надає свою багатофункціональну бібліотеку інстрементів IOS SDK, майже весь функціонал бібліотеки був реалізований у додатку.

Рівень ядра (Core OS) - працює з файловою системою, контролює дійсність різних сертифікатів, що належать додаткам. Також відповідає за безпеку всієї системи в цілому. Містить низький рівень доступу до елементів пристрою.

Надає доступ до таких сервісів як:

— вбудована база даних (SQLite), що дозволяє створювати модель даних, а також здійснювати SQL запити

— файловий диспетчер (File Accesses), сервіс має високий 17 рівень інтерфейсу для доступу до різних файлів належить додатком.

— локація (Core Location) - сервіс, який дозволяє відстежити своє поточне місцезнаходження, а також відстежувати його тривалі переміщення.

— інтернет сервіс (Net Service) - сервіс, який дозволяє отримувати і передавати дані через мережу інтернет [8].

Рівень медіа (Media) - містить інструменти дозволяють обробляти більшість форматів мультимедійних файлів, наприклад:

— аудіо (Core Audio) - дозволяє записувати, прослуховувати, а також і редагувати різні аудіо доріжки. Можливість дістати з відео, певну доріжку.

— відео (Video playback) - сервіс, який дозволяє відтворювати, перемотувати, зупиняти відео. Редагування відео на увазі зміну розширення, довжини або якості.

— анімація (Core Animation) - бібліотека дозволяє створювати анімації. При створенні анімації може використовуватися як показ картинок послідовно, так і зміна різних параметрів елементів інтерфейсу, таких як висота, ширина, розташування, видимість і т.д.

Рівень інтерфейсу (Cocoa Touch) - має безліч елементів для створення мобільних інтерфейсів, а а також надає іншим верствам інформацію вхідну від користувача. У нього входять такі елементи як:

— оброблювач множинних натискань (Multi-Touch) - дозволяє обробляти кілька натискань на екран.

— галерея (ImagePicker) - надає доступ до фотографій і відео лежать на пристрої. За допомогою нього додатки можуть отримати доступ і завантажити до себе файли з галереї.

— оброблювач жестів (Core Motion) - дозволяє обробити різні жести

виробляються на екрані пристрою, такі як зрушення вправо, вниз і т.д.

— камера (Camera) - використовується для знімком і подальшим завантаженням даних в додаток

2.1.2 Огляд моделі веб-серверу

Першим етапом у розробці додатку є визначення стеку взаємодії веб-серверу та інтерфейсу додатку. Обрана мова JavaScript надає можливості створення веб-серверу. При розробці була використана бібліотека Node.js

Node.js використовується для розробки серверної частини проекту. Проте, Node.js також має і можливості для розробки десктопних та мобільних додатків. Node.js використовує модульну систему. Тобто весь її вбудований функціонал розбитий на окремі пакети або модулі. Модуль представляє блок коду, який може використовуватися повторно в інших модулях. При необхідності кожен модуль може бути використаний у будь-якому компоненті [6].

Для роботи з сервером і протоколом http в Node.js використовується модуль http. Щоб запустити сервер потрібен компілятор обробник. Платформа не має вбудованої системи маршрутизації. Через це була використаний фреймворк Express, який налагоджує маршрутизацію та перенаправлення до відповідного контролеру.

Якщо додаток отримуєте великі обсяги необхіднохї інформації, то потрібно правильно підібрати базу даних бо це може стати проблемою Node.js. Рішення в тому, щоб зафіксувати поведінку клієнта перед тим, як дані насправді будуть записані в базу.

При використанні такого підходу чуйність системи зберігається і під високим навантаженням, що особливо корисно, якщо клієнтові не потрібно підтвердження про те, що запис даних пройшла успішно. Кешування даних на стороні веб-серверу запобігає непортібній навазі на сервер та надає можливість отпимізвувати роботу серверу, типові приклади: авторизація

користувача та створення облікового запису. Дані шикуються в чергу за допомогою спеціальної інфраструктури роботи з чергами повідомлень та перетворюються окремим процесом бази даних, призначеним для пакетного запису, або спеціальними сервісами інтерфейсу бази даних, розрахованих на інтенсивні обчислення. Аналогічна поведінка можна реалізувати і за допомогою інших мов фреймворків, але на іншому апаратному забезпеченні і не з такою високою і стабільною пропускною спроможністю.

Node.js цілком можна використовувати в якості серверного проксі, і в такому випадку він може обробляти велику кількість одночасних з'єднань в неблокіруючіх режимі. Це особливо зручно при посередництві між різними сервісами, у яких відрізняється час відгуку, або при зборі даних з багатьох джерел.

2.1.3 Моделювання інтерфейсу

Наступним питанням став вибір платформи для розробки інтерфейсу додатку. Обраною платформою стала NativeScript через те що особливістю вона надає можливість використання усіх інструментів IOS. NativeScript платформа яка потребує фреймворку обгортки за для комфортної розробки інтерфейсу та взаємодії з базою даних [3]. Це бібліотека, що дозволяє робити крос-платформні додатки, використовуючи, CSS, JavaScript. NativeScript вирішує задачу створення крос-платформних додатків, використовує нативний рендеринг, використовує елементи нативного UI. Наступне важлива відмінність: щоб отримати доступ до камери, gps. NativeScript надає доступ з коробки.

Приклад роботи платформи з обгортками, а саме фреймворками зображено на рисунку 2.3

За основу обгортки для платформи NativeScript розробляємого інтерфейсу була взята бібліотека NativeScript-Vue.

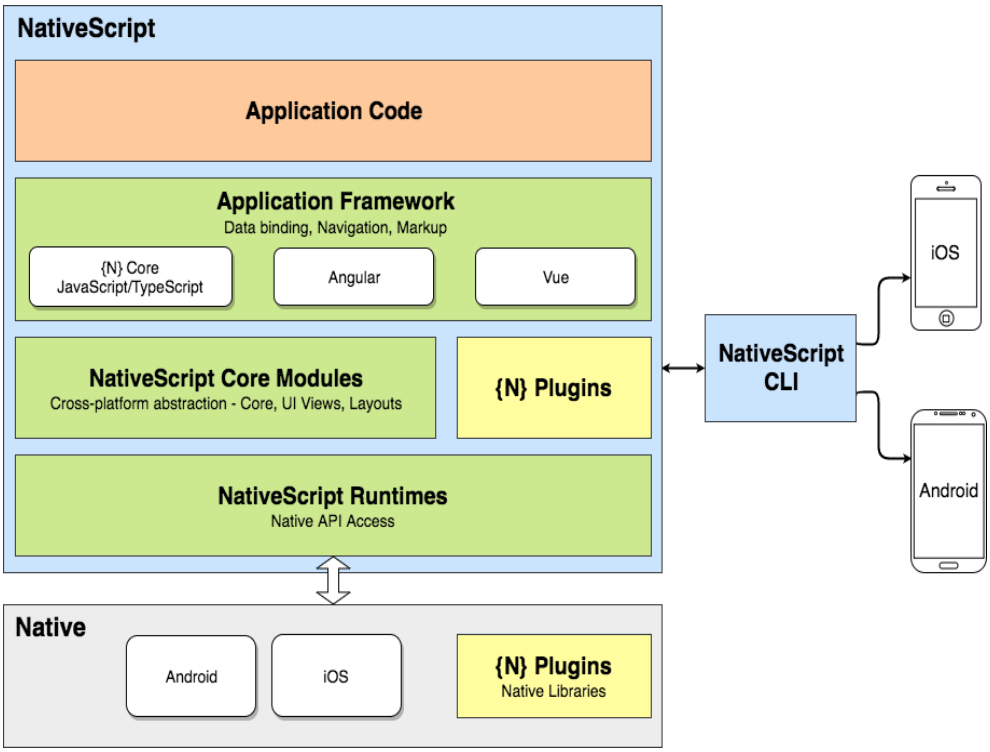

Рисунок 2.3 — Архітектура Native Script

NativeScript-Vue - це плагін для NativeScript, який дає можливість використовувати Vue.js для створення мобільного додатку.

Vue.js - це JavaScript бібліотека для створення веб-інтерфейсів з використанням шаблону архітектури MVVM (Model-View-ViewModel) [4].

Оскільки Vue працює тільки на «рівні уявлення» і не використовується для проміжного програмного забезпечення і бекенда, він може легко інтегруватися з іншими проектами і бібліотеками. Vue.js містить широку функціональність для рівня уявлень і може використовуватися для створення потужних односторінкових веб-додатків

2.2 Розробка моделі додатку

Була розроблена структурна схема взаємодії модулів системи представлена на рисунку 2.4

У додатку реалізована аутентифікації користувача. Кожна платформа має різноманітні процеси та архітектуру, в залежності загальним принципом

роботи аутентифікації є створення JWT токена для поточної сесії користувача.

JSON Web Token (JWT) - це JSON об'єкт, який визначений у відкритому стандарті RFC 7519. Він вважається одним з безпечних способів передачі інформації між сервером та користувачем. Для його створення необхідно визначити заголовок (header) із загальною інформацією по токені.

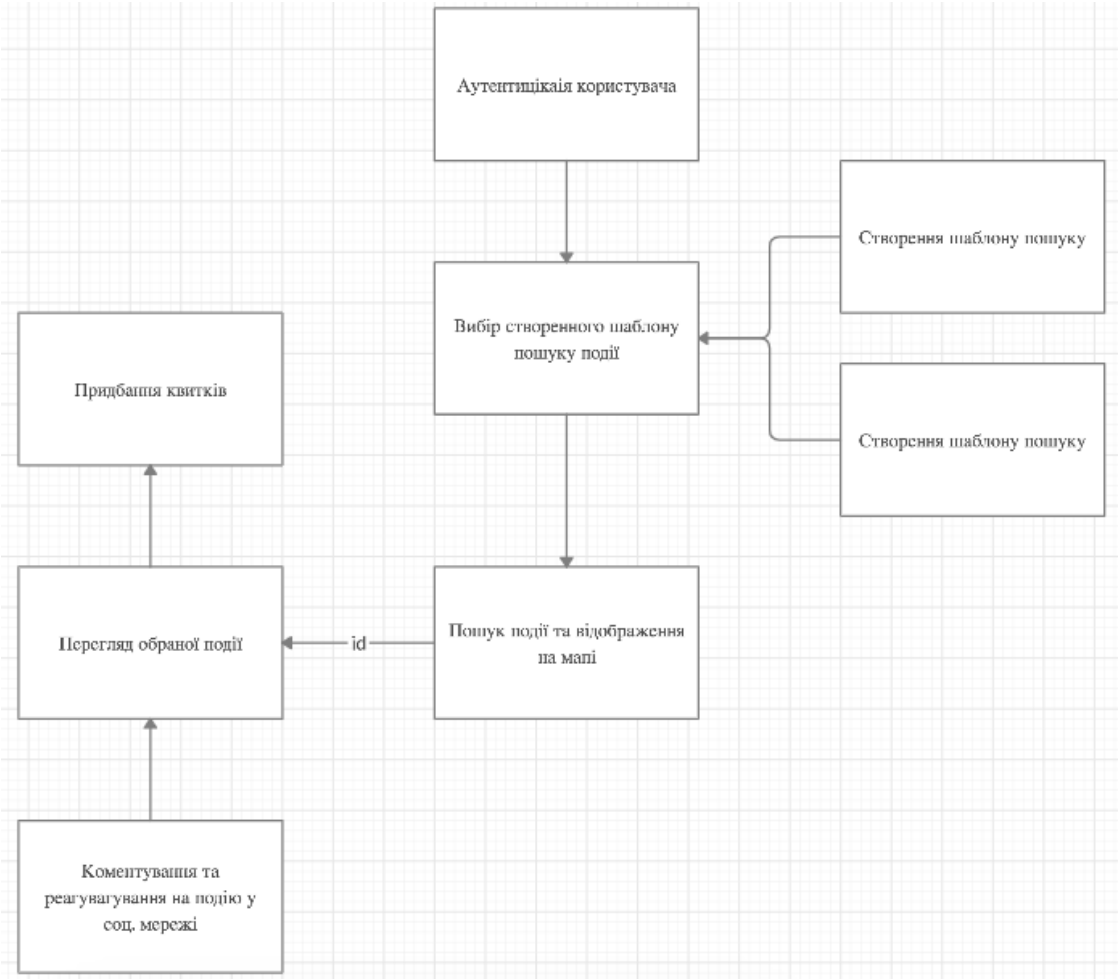

Рисунок 2.4 — Взаємодія модулів системи

Поетапна структура аутентифікації користувача:

— користувач звертається до серверу аутентифікації за допомогою форми логіну або Facebook ключа;

— потім сервер аутентифікації створює JWT і відправляє його користувачеві;

— якщо токени користувача та серверу співпадають тоді користувач входить у систему;

— коли користувач робить запит до API додатка, він додає до нього отриманий раніше JWT.

Реалізація аутентифікації користувача до системи наведено на рисунку 2.5

Рисунок 2.5 — Аутентифікація користувача

Наступний етап взаємодії користувача та додатку це створення шаблону пошуку подій. Основною метою цієї функції є створення пошукової системи за своїм бажання Принцип роботи:

— заповнення форми пошуку;

— відправка форми до веб-серверу;

— обробка та створення нового шаблону пошуку;

— запис шаблону до таблиці бази даних.

Діаграма роботи створення шаблону пошуку представлена на рисунку 2.6

Рисунок 2.6 — Діаграма створення шаблону пошуку

Основний екран додатку представляє собою мапу із зображенням місцезнаходження користувача та подій згідно шаблону пошуку. Кожен маркер події створюється згідно даних з запиту та відображається поблизу користувача. Основні принципи роботи даного етапу:

— запит до веб серверу з поточним фільтром пошуку;

— надання інформації відносно місцезнаходження користувача;

— можливість перегляду детальної інформації кожної події;

Схема взаємодії мапи та користувача представлена на рисунку 2.7

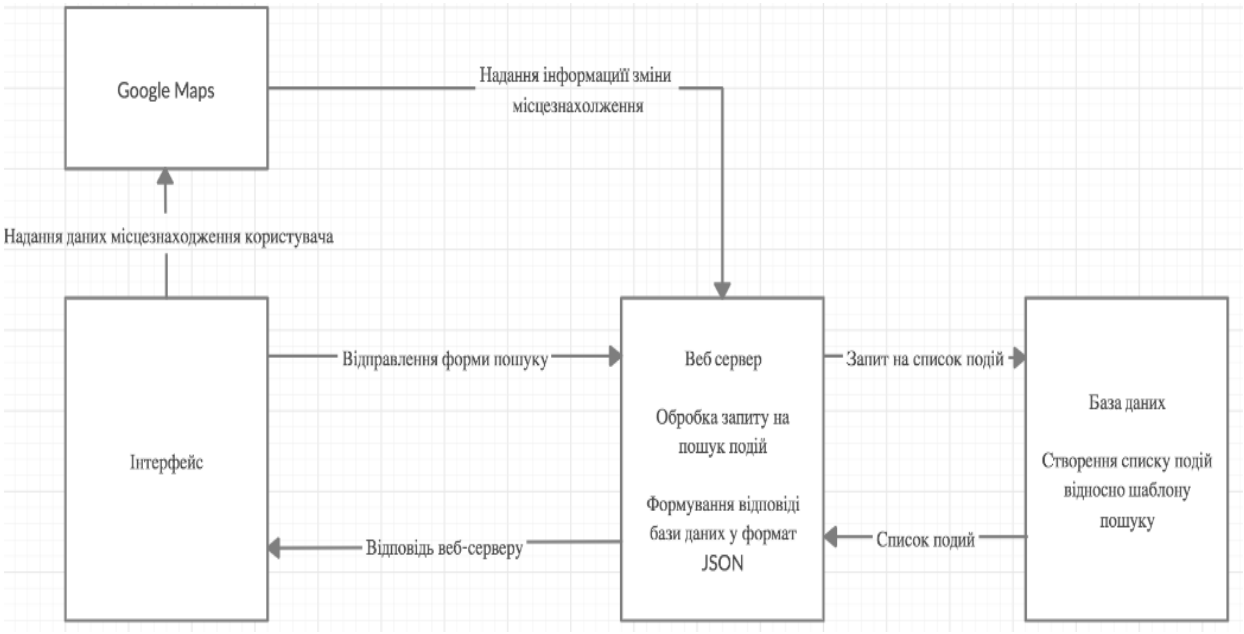

Рисунок 2.7 — Схема взаємодії мапи

2.3 Моделювання інтеграції геолокації користувача

Інтеграція Google Maps SDK. Щоб розробити мобільний додаток з геолокації, потрібні послуги з визначення місця розташування і карти. Вони додаються за допомогою API - набору правил для взаємодії додатків з операційними системами. Для кожної ОС розроблений свій набір API. Для використання Core Location необхідно підключити CoreLocation.framework в до Xcode додатку . Для отримання доступу до класів і заголовкам, включите import CoreLocation / CoreLocation.h в початок файлу реалізації, використовує Core Location. Також, починаючи з iOS 8, файл Info.plist повинен містити ключ NSLocationWhenInUseUsageDescription зі строки запиту на дозвіл використання служб геолокації для додатка.

Core Location дозволяє отримати ваше місце розташування від пристрою і використовувати її у додатку. Існує декілька варіантів настройки послуги визначення місцезнаходження, пов'язані з:

— використанням стандартної служби визначення місцеположення, яка дозволяє вказати потрібну точність даних про місцезнаходження і отримувати оновлення при зміні місця розташування;

— використанням значущою служби зміни розташування, яка забезпечує більш обмежений набір опцій стеження, але пропонує значну економію електроенергії в порівнянні зі стандартною службою визначення місцезнаходження. В iOS, ця служба також може запустити програму при необхідності доставки оновлення розташування.

Інтеграція Google Maps SDK for IOS. Стандартна служба визначення місцезнаходження є найбільш поширеним способом отримати поточне місце розташування користувача, тому що вона доступна на всіх пристроях і iOS і OS X. Точність стандартної служби визначення місцеположення потрібно навігаційним додатків або будь-якого додатка, яке вимагає високої точності визначення даних місцезнаходження або регулярному потоці оновлень.

Оскільки ця служба зазвичай вимагає тривалого включення апаратних засобів визначення місцеположення, це може призвести до більш високого енергоспоживання.

За допомогою SDK Maps for iOS надає можливість додавати карти на основі даних карт Google у свою програму. SDK автоматично обробляє доступ до серверів Карт Google, відображення карти та відповідь на жести користувачів, такі як кліки та перетягування. На свою карту можна також додати маркери, полілінії, наземні накладки та інформаційні вікна. Ці об'єкти надають додаткову інформацію про розташування карт та дозволяють взаємодію користувачів із картою.

Екран детального перегляду обраної події виконує такий функціонал:

— перегляд усієї інформації про обрану подію;
— можливість додавати подію до бажаних;
— коментування події через соціальну мережу;
— придбання квитків.

2.4 Моделювання інтеграції соціальних мереж.

Кожна подія інтегрована з соціальною мережею Facebook, котра надає можливість спілкування користувачів на тему події. Робота екрану взаємодії поточної події з соціальною мережею та платіжним методом представлена на рисунку 2.8.

Інтеграція соціальної мережі - це одна з основних інтеграцій в будь-якому мобільному додатку. Воно змінює спектр використання програми на інший рівень. Обраною соціальною мережею була - Facebook. Ця платформа надає великий інструментарій SDK Facebook, який містить увесь спектр потрібних функцій для розробки додатку.

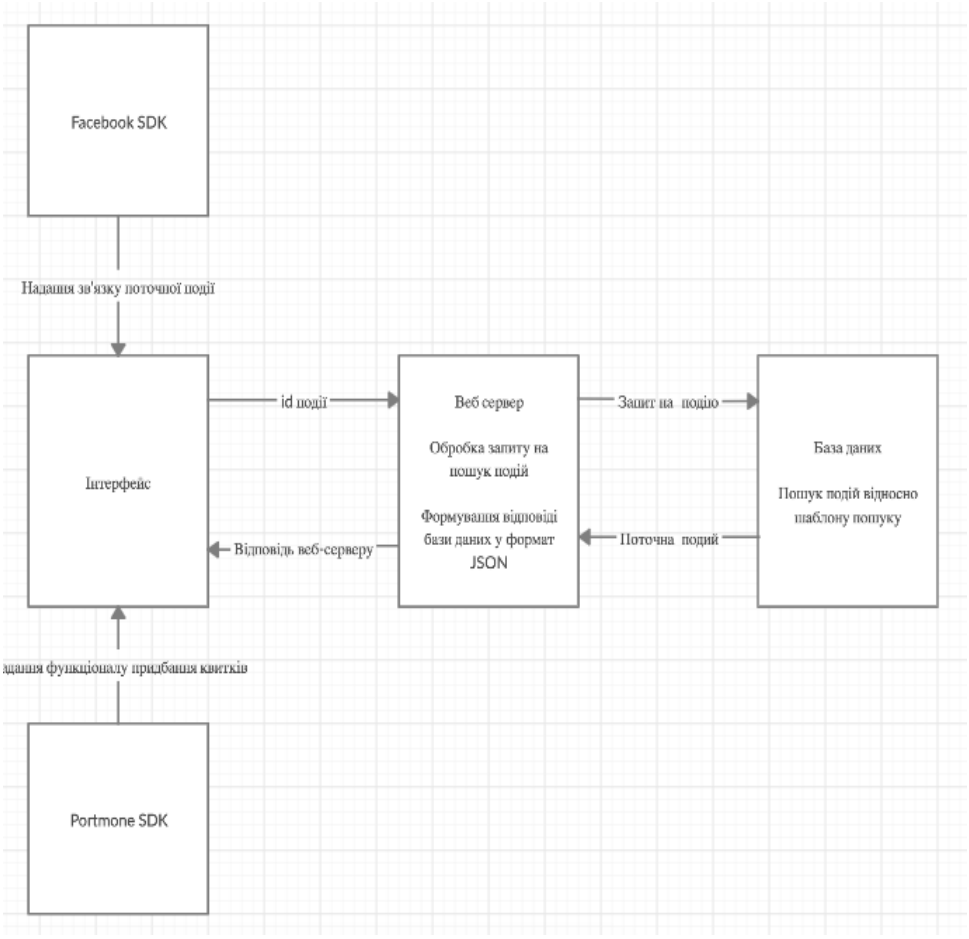

Рисунок 2.8 — Модель взаємодії соц. мережі та платіжної системи

Під час використання SDK Facebook певні події у програмі автоматично реєструються та збираються для Facebook Analytics. Ці події актуальні для всіх випадків використання - націлювання, вимірювання та оптимізація. API Graph - це основний інструмент для завантаження даних на платформу Facebook і їх отримання звідти. Він являє собою API на базі HTTP, за допомогою якого додатки можуть програмним шляхом запитувати дані, публікувати нові історії, управляти рекламою, завантажувати фото і виконувати безліч інших завдань.

Назва API Graph підкреслює зв'язок цього API з «соціальним графом» - системою подання інформації на Facebook. API Graph складається з наступних елементів:

— вузли - окремі «об'єкти» (наприклад, Користувач, Фото, Сторінка або Коментар);

— кордону контексту - зв'язку між підбіркою об'єктів і окремим об'єктом, наприклад фото на Сторінці або коментарі до фото;

— поля - дані про об'єкт, наприклад день народження користувача або назву Сторінки.

За допомогою вузлів можна отримувати дані про конкретний об'єкт, межі контексту дозволяють збирати добірки об'єктів, пов'язані з окремим об'єктом, а поля допомагають отримувати дані про окремий об'єкт або про кожен об'єкт в добірці.

Отже, інтеграція соціальної мережи дуже проста, API Graph має дуже гнучке налаштування взаємодії через Api запити. На рисунку 2.9 привидений приклад відправки коментарів щодо поточної події.

```
axios.method(
  '/{event_id}/comments',
  'POST',
  {"message":"Awesome!"},
  function(response) {
      this.list = responce.data
  }
);
```

Рисунок 2.9 — Запит створення коментарія

2.5 Моделювання інтеграції платіжних систем.

Інтеграція платіжної системи - це грошові операції, які здійснюють за допомогою мобільного пристрою, найчастіше смартфона, тобто всі платежі відбуваються без використання готівки і кредитних карт.

Однією з ключових переваг мобільних платежів - вони можуть виконуватися як в режимі офлайн, коли користувачеві необхідно віднести

мобільний пристрій до безконтактного платіжного терміналу, так і онлайн - здійснення покупок в додатках і безпосередньо в мережі Інтернет.

Мобільні способи оплати, запропоновані Google, Apple і Samsung, більш безпечні, ніж платіжні картки і готівка. Міжнародні платіжні системи для мобільних пристроїв використовують такі методи захисту даних, як шифрування і токенізація. Це дозволяє маскувати дані платіжних карт при здійсненні оплати.

Apple Pay - платіжна система Apple дозволяє здійснювати оплату та спілкуватися на веб-сайтах та мобільних додатках без прямого зв'язку. Одержувачу платежу потрібно лише підключити свою карту та підтвердити інші платежі за допомогою TouchID або FaceID. Ця технологія доступна власникам пристроїв Apple (браузери Safari на iPhone, iPad та деяких моделях MacBook). Інтеграційна платформа була використана для розробки програм [13]. Не потрібно додаткової інтеграції з цим методом з'єднання. Кнопка Apple Pay з’явиться на сторінці платежів Portmone.com. Способи оплати можна використовувати для перевірки та перегляду платіжних інструментів (включаючи Apple Pay) за допомогою запиту JSON.

Таблиця 2.1 – Параметри запитів Portmone

Параметр	Опис	Обов’язковий
login	Логін для доступу до управління аккаунтом	Так
password	Пароль Інтернет-магазину	Так
aPayMerchantName	Айді користувача	Так
paymentData	Значення параметру payment Data, що отриманий мерчантом у відповіді від системи Apple Pay	Так

Параметр	Опис	Обов’язковий
bilAmount	Сума оплати	Так
description	Коментар до замовлення	Ні
shopOrderNumber	Номер замовлення (рахунку)	Ні
emailRecipient	Адреса електронної пошти Клієнта	Ні
preauthFlag	Ознака передоплати платежу	Ні
billCurrency	Валюта проведення платежу	Ні
shop_site_id	Канал оплати у системі Portmone	Ні

Таблиця 2.2 –Параметри відповідей Portmone

Параметр	Опис
status	Результат оплати
errorCode	Код помилки.
error	Текст помилки
shopBillid	Ідентифікатор транзакції (платіжного документу) у системі Portmone.com
billAmount	Передана у запиті сума транзакції
billNumber	Номер замовлення
attribute	Службове поле
cardMask	Маска Картки / Token плтаника

Параметр	Опис
actionMPI	url банку
pareq	Параметр, який треба передати на actionMPI при проходженні 3D Secure перевірки
authCode	Код авторизації
description	Коментар до замовлення
lang	Мова відповіді
md	Параметр, який треба передати на actionMPI при проходженні 3D Secure перевірки
token	Значення Токену для подальших оплат
shop_site_id	Канал оплати у системі Portmone

Для більш точної орієнтації роботи платіжної системи у розробляемому додатку був розглянутий списко методів надаваємий платіжним методом за для створення архитектури взаємодії додатку та платіжної системи. Архітектура роботи запитів наведено на рисунку 2.10.

Portmone.com є продукт, орієнтований на сегменти P2B (моментальні і регулярні платежі) і B2B (платформа для впровадження сервісу по прийому платежів). Платформа має надійну та просту у використанні інтеграція для мобільних додатків.

Виходячи з вище розроблених модулів у системі повинен бути реалізований такий функціонал:

— аутентифікація користувача;

— створення шаблону пошуку подій;

— інтеграція Google Maps у додаток;

— відображення місцезнаходження користувача;

— відображення подій;

— інтеграцію соціальної мережі;

— реалізацію коментарів та відгуків для кожної події;

— підключення платіжної системи у додаток.

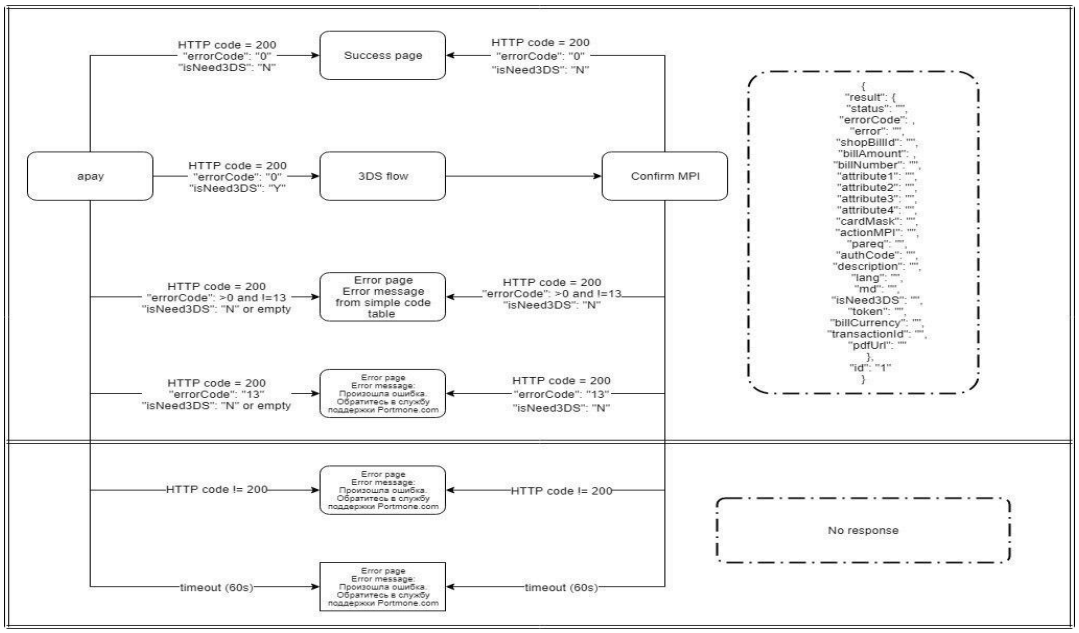

Рисунок 2.10 –Архітектура роботи запитів платіжної системи

Висновки до розділу

В даному розділі була створена структура взаємодії модулів системи, модулі системи:

— авторизація з використання JWT токену користувача;

— структура створення шаблону пошуку подій;

— інтерфейс мапи з використання бібліотеки GoogleMaps;

— модуль роботи соціальної мережі Facebook

— модуль роботи платіжної системи Portmone

Проведений аналіз та вибір необхідних бібліотек до кожного модулю системи. Також розроблений список задач основного функціоналу додатку з використання зовнішніх бібліотек.

3 МОДЕЛЮВАННЯ ТА КОНСТРУКЦІЯ МОБІЛЬНОГО ДОДАТКУ

Метою розробляємого додатку є створення сучасної платформи пошуку цікавих користувачу подій. Програма надає можливість у режимі реального часу відстежувати події на мапі відносно місцезнаходження користувача.

Додаток поділяється на базові екрани:

— Login - екран відповідає за авторизацію користувача у систему.

— Features - заповнення форми бажаних користувачу події.

— Events- відображення місцезнаходження події.

— Event - відображає детальну інформацію події.

— Payment - екран придбання квитків.

3.1 MVVM и Coordinators в iOS приложениях

Реалізація паттерну MVVM необхідна для правильної архітектури розробки додатку. Притримуючись цієї архітектури додаток можливо оновлювати в будь-який час без проблем у розробці, усі модулі системи працюють незалежно.

3.1.1.MVVM

Шаблон дозволяє розділити додаток на три функціональні частини:

— Model - логіка програми (робота з даними, обчислення, запити до бази даних).

— View - вид або подання (призначений для користувача інтерфейс).

— ViewModel - модель уявлення, яка служить зв’язком між View та Model.

Такий поділ дозволяє прискорити розробку та оновлення програми. Також MVVM додає абстракцію View - ViewModel, яка займається відстеженням змін в даних моделі та їх відображенням уявлення , такий принцип був реалізований як на веб-сервері так і в інтерфейсі. Принцип роботи MVVM зображено на рисунку 3.1

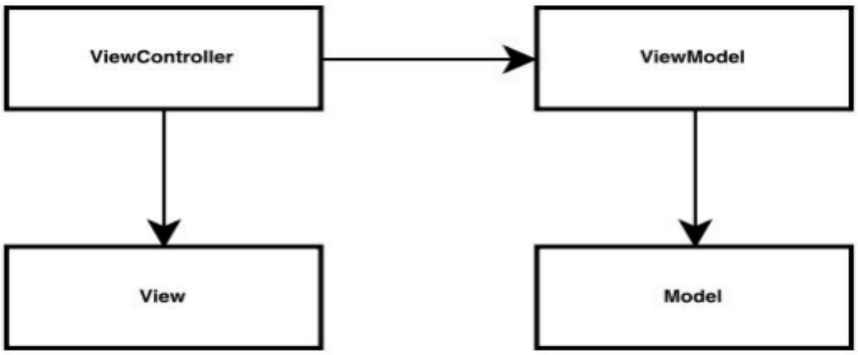

Рисунок 3.1 — Модель MVVM в iOS додатку

MVVM застосовується для поділу моделі і її уявлення, що необхідно для зміни їх окремо один від одного. наприклад, розробник працює з логікою даних, в той час дизайнер відповідно розробляє призначений для користувача інтерфейс. MVVM зручно використовувати замість класичного MVC і йому подібних в тих випадках, коли в платформі, на якій ведеться розробка, присутня явна зв'язаність між призначеним для користувача інтерфейсом і бізнес логікою [10].

3.1.2 Паттерн Coordinator

Розробка навігаційної системи додатку потребує зовнішньої бібліотеки. Coordinator (Координатор) - паттерн дозволяє поєднувати взаємодію навігації серверу та двига платформи IOS. Координатор може управляти також координаторами, доданими до головного батьківського

координатору, така можливість була використана при відображенні координат користувача у модулі мапи .

Розглянемо основний інтерфейс координатора. Архітектура взаємодії методів паттерну Coordinator представлена на рисунку 3.2

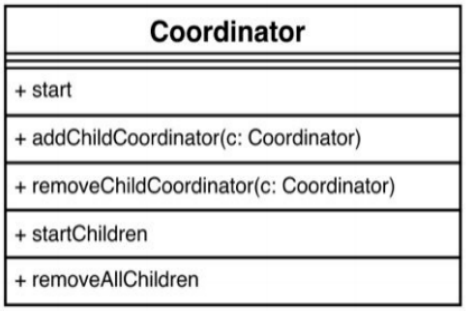

Рисунок 3.2 — Інтерфейс координатора

Зазвичай додаток стартує з головного координатора, який в залежності від стану покаже необхідний додатком координатор.

Наявний приклад реалізації патерну Coordinator у додатку представлений на рисунку 3.3

```
class ApplicationCoordinator: Coordinator {
  let kanjiStorage: KanjiStorage //  1
  let window: UIWindow  // 2
  let rootViewController: UINavigationController  // 3

  init(window: UIWindow) { //4
    self.window = window
    kanjiStorage = KanjiStorage()
    rootViewController = UINavigationController()
    rootViewController.navigationBar.prefersLargeTitles = true

    // Code below is for testing purposes   // 5
    let emptyViewController = UIViewController()
    emptyViewController.view.backgroundColor = .cyan
    rootViewController.pushViewController(emptyViewController, animated: false)
  }

  func start() {  // 6
    window.rootViewController = rootViewController
    window.makeKeyAndVisible()
  }
}
```

Рисунок 3.3 — Основний координатор додатку

3.2 Використані інструменти розробки додатку.

Xcode Simulator - інтегроване середовище розробки iOS. Додаток створений у емуляторі, який надає велику бібліотеку функціоналу. Застосований інструмент являє собою написання програми в симуляторі на будь-якій базі програмного забезпечення, що оптимізує робочий процес розробки [12]. Підключення симулятору надає середовище розробки Xcode приклад запуску симулятора зображено на рисунку 3.4

Рисунок 3.4 — Запуск IOS симулятора при прозробці додатку

Інтерфейс IOS Simulator представлено на рисунку 3.5

Рисунок 3.5 — IOS Simulator

Для розробки додатку були розглянуті два конкуруючі платформи які надають можливості емулятора додатку IOS та Android. В таблиці 3.1 були розглянуті основні переваги та недоліки платформ.

Таблиця 3.1 –Порівняння симуляторів IOS та Android

Крітерії	Платформи	
	IOS	Android
Якість графічного дизайну	+	-
Налаштування девайсу під кожного юзера	-	+
Регулярність оновлення ПО	+	-
Повна функціональність смартфону	+	-
Простота підключення	-	+
Екосистема продукції	+	-
Фасування даних між продукцією	+	-

Виходячи з аналізу таблиці платформа IOS має значні переваги у функціональній галузі. Однак обидві операційні системи ж кожним часом додають все більше можливостей реально смартфону. IOS, в свою чергу,

працює над функціональністю та можливістю індивідуального налагодження пристрою.

3.3 База даних

Важливою темою в розробці додатків є вибір найбільш підходящої бази даних. Основними критеріями відбору є здатність бази даних взаємодіяти як звичайний клас. База даних - це структура взаємопов'язаних таблиць з великою кількістю параметрів, які взаємодіють між собою. Взаємодія всіх таблиць баз даних утворює незалежну структуру архітектури розробленого додатку і складає безліч варіацій інформаційного групування.

Такі системи відрізняються обробкою даних та централізацією даних. Найпопулярніша система управління - MySQL. Це забезпечує простий доступ до управління базами даних. Система управління бази даних надає весь необхідний функціонал для розробки мобільного додатку та забеспечує роботу веб-серверу з усіма необхідними зв’язками між таблицями.

Основна перевага бази даних - швидкість оновлення та використання необхідної інформації, алгоритми якої надає система управління MySQL. [14]

Наявний приклад реалізації підключення бази даних до розробляємого додатку наведено на рисунку 3.6

```
module.exports = {
  port: process.env.PORT || 8081,
  db: {
    database: process.env.DB_NAME || 'p&g',
    user: process.env.DB_USER || 'p&g',
    password: process.env.DB_PASS || 'p&g',
    options: {
      dialect: process.env.DIALECT || 'mysql',
      host: process.env.HOST || 'localhost',
      storage: './p&g.sqlite'
    }
  },
  authentication: {
    jwtSecret: 'process.env.JWT_SECRET' || 'secret'
  }
}
```

Рисунок 3.6 — Підключення бази даних

В наведеному прикладі реалізоване підключення бази даних. Задані усі можливі налаштування:

— назва бази даних;

— ім’я володаря бази даних;

— пароль володаря;

— обрана платформа;

— заданий поточний хост розробки додатку;

— файл збереження помилок компіляції;

— підключення JWT токена.

На рисунку 3.7 зображено звертання моделі до бази даних за для створення користувача.

```
module.exports = (sequelize, DataTypes) => {
  const User = sequelize.define('User', {
    email: {
      type: DataTypes.STRING,
      unique: true
    },
    password: DataTypes.STRING
  }, {
    hooks: {
      beforeCreate: hashPassword,
      beforeUpdate: hashPassword,
      beforeSave: hashPassword
    }
  })
```

Рисунок 3.7 — Створення нового користувача

Вхідні параметри формуються у рядки бази даних із заданими типами даних які будуть записані до таблиці. Життєвий цикл створення запису перевіряє наявність валідного паролю для забезпечення безпеки додатку.

СУБД - це програмне забезпечення, яке виступає посередником між базою даних та її користувачами. За допомогою мовних та програмних засобів база даних полегшує створення, обслуговування та обмін базами даних різними користувачами. Обрана система управління MySQL є лідером

у використанні швидко працюючих додатків за своєю уникальною архітектурою.

Сучасна програма СУБД складаються з ядра, процесору мови БД, підсистеми підтримки часу виконання та сервісних програм, які надають додаткові можливості для обслуговування інформаційних систем. Також СУБД нажає можливість коректоного відстеження версії бази та встановлення необхідних пакетів розробки для більш швидкої або зручної роботи БД.

Використання звертання до бази даних через модель сущності додатку User зображно на рисунку 3.8

Рисунок 3.8 — Звертання до бази даних

Модель додатку представляє собою чітке поняття які властивості потребує таблиця бази даних. Модель Users має 2 вхідних параметри:

— email;

— password.

Загальна схема бази даних представлена на кресленику ІА-62.

3.4 Розробка веб-серверу

Основна взаємодія додатку походить з основи розробки веб-серверу, обробки даних та за допомогою HTTP-протоколів. Протоколи безпосередньо відправляються на сервер, сервер шукає дані, вибудовує їх в шаблон, а потім повертає у вигляді HTML-сторінки. Між отриманням запиту і відповіддю на нього сервер зазвичай шукає по сформованому запиту інформацію в БД [15].

Кожен запит/відповідь складається з трьох частин:

— стартовий рядок;

— заголовки;

— тіло повідомлення, що містить дані запиту, запитаний ресурс або опис проблеми, якщо запит не виконано.

Заголовки несуть в своєму контексті опціональну інформацію про запит, аутентифікацію користувача, формат запиту, кодування. Тіло повідомлення несе параметри запиту до БД для отримання потрібної інформації.

Формат працює на основі взаємодії ключа та значення, який зазвичай рендерится в фігурних дужках. Робота з JSON інтерпретує об'єкти в .json файл, але вони також можуть бути і як JSON об'єкт або рядок вже в контексті самої програми.

Сервер і додаток обмінюються інформацією у вигляді текстового формату JSON. Формат відповіді на запит представлено на рисунку 3.9.

Протокол звертається до контролеру, який обробляє параметри з реквесту та створює новий запит до БД. Звертання до БД налаштовується в залежності від платформі. у якій проходить розробка додатку.

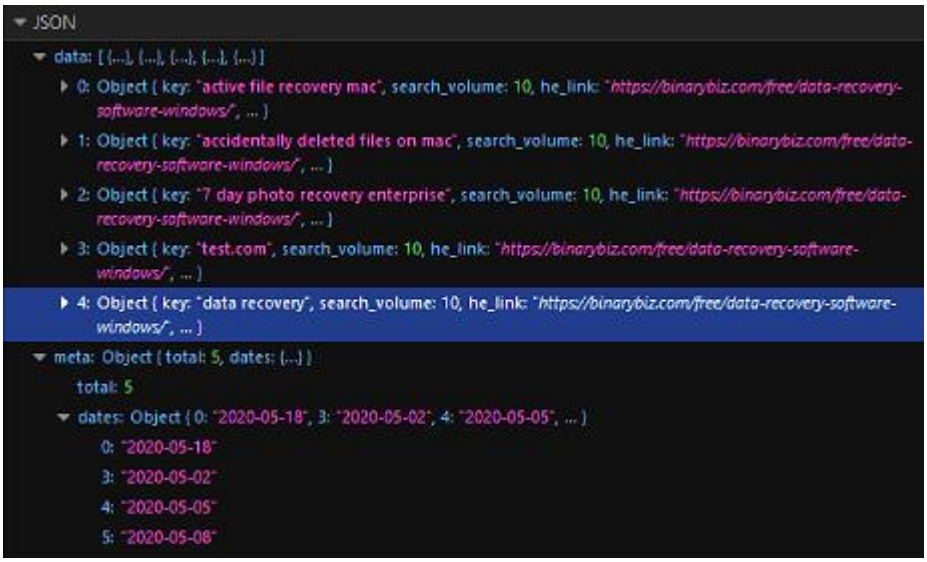

Рисунок 3.9 — Відповідь HTTP запиту

Приклад роботи контролера авторизації користувача у систему представлено на рисунку 3.10

```
async login (req, res) {
  try {
    const { email, password } = req.body
    const user = await User.findOne({
      where: {
        email: email
      }
    })
```

Рисунок 3.10 — Авторизація користувача у контролері

Контролер приймає вхідні дані та обробляє у формат запиту до бази даних. Параметр where означає критерій пошуку у таблиці User. Якщо пошук користувача невдалий то відповідь запиту представлена на рисунку 3.11.

```
if (!user) {
  return res.status(403).send({
    error: 'Check your login please'
  })
}
```

Рисунок 3.11 — Відповідь веб-серверу на невдалий пошук користувача

Сервер поверне помилку 403, що означає невдалу авторизацію. Також після перевірки статусу запиту контролер перевіряє валідність вхідних даних зображено на рисунку 3.12

```
const isPasswordValid = await user.comparePassword(password)
if (isPasswordValid) {
  return res.status(403).send({
    error: 'Password is invalid for this email'
  })
}
```

Рисунок 3.12 — Перевірка валідності параметрів запиту

Кожний веб-сервер потребує чітку структуру пакетного менеджеру, який відповідає за оновлення та завантаження усіх бібліотек додатку, це - npm (node package manager).

3.4.1Пакентий менеджер Npm

Npm (node package manager) - це менеджер пакетів Node.js. У першій частині цього матеріалу ми вже згадували про те, що зараз в npm є понад півмільйона пакетів, що робить його найбільшим в світі репозиторієм коду, написаного на одній мові. Це дозволяє говорити про те, що в npm можна знайти пакети, призначені для вирішення практично будь-яких завдань.

Спочатку npm створювався як система управління пакетами для Node.js, але в наші дні він використовується і при розробці фронтенд-проектів на JavaScript. Для взаємодії з реєстром npm використовується однойменна команда, яка дає розробнику величезна кількість можливостей. Кожен веб-сервер повинен мати файл формату package.json, який формує структуру налаштування та інсталяцію взаємодії бібліотек node.js. npm зберігає усі залежності та встановлені пакети до єдиного файлу package.json [16]. Використані та пакети веб-серверу у додатку зображено на рисунку 3.13.

Рисунок 3.13 — Встановленні та використанні залежності npm

Приклад вхідних параметрів npm представлено на рисунку 3.14

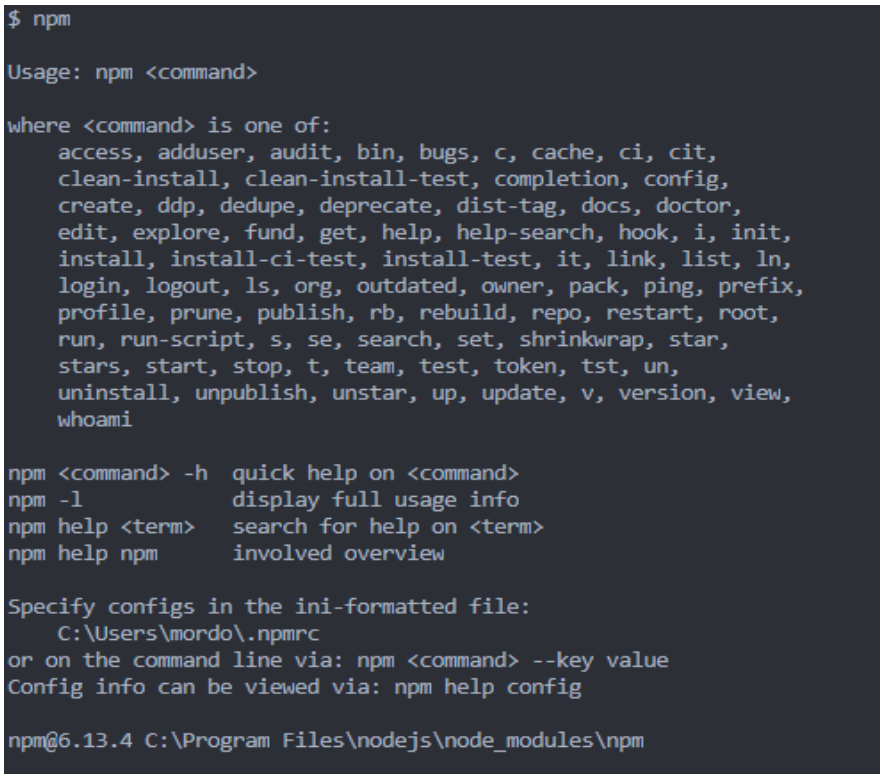

Рисунок 3.14 — Налагодження пакетного менеджеру

Основна команда інсталяції є “npm install”. Додаток до стандартної завантаженні пакетів, npm підтримує і завантаження їх певних версій.

Встановлення необхідних залежностей у додатку представлено на рисунку 3.15

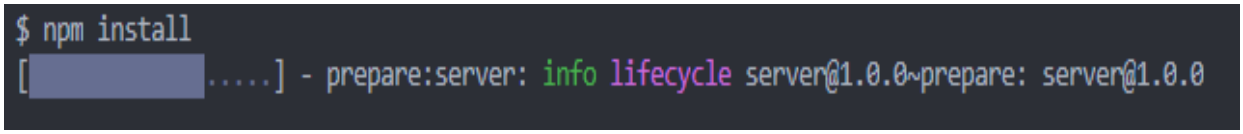

Рисунок 3.15 — Встановлення залежностей

Зокрема, можна помітити, що деякі бібліотеки сумісні лише з якимись великими релізами інших бібліотек, тобто, якби залежності таких бібліотек встановлювалися б без урахування версій, це могло б порушити їх роботу. Можливість встановити точну версію якогось пакета корисна і в ситуаціях, коли, наприклад, вам цілком підходить найсвіжіший реліз цього пакета, але виявляється, що в ньому є помилка. Чекаючи виходу виправленої версії пакету, можна скористатися і його старішим але стабільним релізом

Файл package.json має інструкції запуску додатку, які зображені на рисунку 3.16.

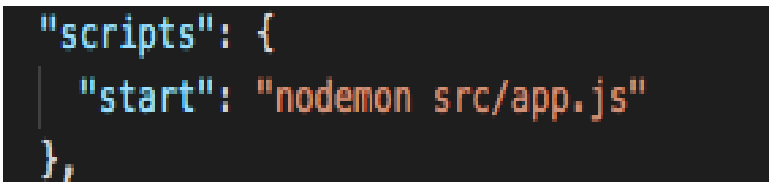

Рисунок 3.16 — Команда запуску веб-серверу додатку

3.4.2 Структура веб-серверу

Express - це мінімалістичний і гнучкий веб-фреймворк для додатків Node.js, що надає великий набір функцій для мобільних і веб-додатків.

Фреймворка не має строгої типізації для структури оформлення проекту, отже приклад архітектури середовища веб-серверу представленно на рисунку 3.17.

Кожна з наведених папок несе свій функціонал:

— /app - папка відповідальна за двосторонню обробку даних з реквесту та формування запиту до Бази даних. Основна логіка роботи веб-серверу налаштовується у цій папці.

— /bootstrap - папка формує основні моделі класів, які мають параметри та залежності. Моделі використовуються скрізь де йде звертання до Бази даних та усіх можливих звертань до моделей.

— /resources - папка відповідає за основне спілкування представлень з контролерами, конфігурація звертання до потрібного контролеру з інтерфейсу.

— /database - конфігурації та міграції бази даних.

— /routes - модуль відповідальний за навігацію додатку.

— /config - папка з налаштуваннями роботи веб-серверу.

— /public - містить побудований додаток інтерфейсу

— /tests - папка у якій розгортаються тести роботи веб-серверу додатку

Рисунок 3.17 — Структура архітектури веб-серверу

Використання MVVM паттерна використовується в усіх контролерах веб-серверу. На прикладі контролеру Events зображеного на рисунку 3.18 розглянемо отримання списку подій.

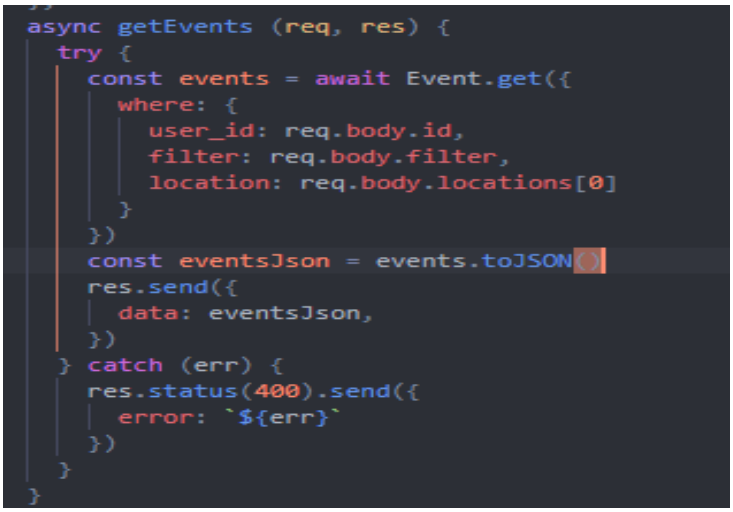

Рисунок 3.18 — Метод getEvents контроллеру Event

Метод визивається кожен раз при потребі оновлення чи отримання спискую подій. Звертається до підключеної моделі Events, яка в свою чергу створює запит до бази даних, також у методі формуються параметри пошуку сущностей:

— user_id, айди користувача;

— filter, створений шаблон пошуку подій;

— location, місцезнаходження користувача.

Модель Events представляє собою усі параметри сущності події зображених на рисунку 3.19.

Отриманий список подій з бази даних формується у Json формат та відправляється у відповідь отриманому запиту.

Файлі app.js та server.js - відповідають за зв’язок серверу та інтерфейсу додатку, налаштування прокси та HTTP запитів.

Статус коди вказують на результат HTTP запиту:

— 1XX - інформаційний;

— 2XX - успішне виконання;

— 3xx - перенаправлення;

— 4xx - помилка клієнта;

— 5xx - помилка сервера.

```
module.exports = (sequelize, DataTypes) => {
  const Events = sequelize.define('Event', {
    name: {
      type: DataTypes.STRING,
      unique: true
    },
    type: DataTypes.STRING,
    cost: DataTypes.Number,
    user_id: DataTypes.INTEGER,
    location: DataTypes.ARRAY(DataTypes.INTEGER),
  })

  return Events
```

Рисунок 3.19 — Сущності моделі Events

Підключення платіжної системи у додаток базується на API токені. Налаштування проходить у два етапи:

— генерація апі токена;

— створення та налаштування контролера;

Контролер платіжного методу представлено на рисунку 3.20

```
const createStoreProducts = async () => {
  try {
    const stripeProducts = await Promise.all(
      products.map(async product => {
        const stripeProduct = await stripe.products.create({
          id: product.id,
          name: product.name,
          type: 'good',
          attributes: Object.keys(product.attributes),
          metadata: product.metadata,
        });

        const stripeSku = await stripe.skus.create({
          product: stripeProduct.id,
          price: product.price,
          currency: config.currency,
          attributes: product.attributes,
          inventory: {type: 'infinite'},
        });

        return {stripeProduct, stripeSku};
      })
    );

    console.log(
      `🛍  Successfully created ${stripeProducts.length} products on your Stripe account.`
    );
  } catch (error) {
    console.log(`⚠  Error: ${error.message}`);
  }
```

Рисунок 3.20 — Контролер платіжної системи

Висновки до розділу

У розділі описано розробку мобільного додатку, а саме:

— авторизація користувача;
— створення шаблону пошуку події;
— підключення інструменту платіжної системи;
— інтеграція соціальної мережі;
— реалізація паттерна MVVM;
— підключення бази даних;
— архітектуру папок веб-серверу;
— використання пакетного менеджеру.

Налагодження та використання інтегрованого інструменту IOS Simulator. Проведений аналіз емуляторів платформ та виявлення загальних переваг платформи IOS над Android, реалізовані патерни розробки MVVM для правильної взаємодії між веб-сервером та інтерфейсом. Була розглянута платформа IOS та використаний інструмент IOS Coordinator для поєднання навігації веб-серверу та платформи. Розглянута робота пакетних менеджерів, налагодження та встановлення залежностей. Обрана мова надає можливість підійти до питання баз даних, які були розглянуті та виявлені основни потреби роботи з нею, за основу була взята база даних - MySQL через простоту у використанні та наявності бібліотеки Sequelize, яка надає можливість звертатися до бази даних як до классу. Була обрана найвигідніша СУБД для правильної архітектури взаємодії таблиць бази даних. Збудована архітектура взаємодії з базою даних на основі методів класу.

Розробка веб-серверу на платформі Node.js передбачає максимально просту архітектуру проекту, були розглянутий приклади із розробленого проекту. Розроблене Api працює на HTTP запитах, тож було детально розглянуто роботу запитів.

4 РОЗРОБКА ВЕБ-ІНТЕРФЕЙСУ ДОДАТКУ

Основним питанням при розробці інтерфейсу був правильний вибір платформи для розробки. Як вже відомо фреймворк NativeScript надає можливості використання інструментів мобільного додатку IOS.

Функції Vue.js:

- реактивні інтерфейси;
- декларативний рендеринг(ssr);
- зв'язування даних;
- директиви;
- логіка шаблонів;
- компоненти;
- обробка подій;
- властивості;
- переходи і анімація CSS;
- фільтри.

Vue підходить для невеликих проектів, яким необхідно додати трохи реактивності, уявити форму за допомогою AJAX, відобразити значення при введенні даних користувачем, авторизація або інші аналогічні завдання. Vue легко масштабується і добре підходить для об'ємних проектів, тому його називають прогресивним фреймворком. Хоч Vue і не реалізує паттерн MVVM в повній мірі, архитектура фреймворка їм багато в чому надихнула. Тому змінну з екземпляром Vue традиційно називають vm (скорочено від ViewModel).

Кожен екземпляр Vue при створенні проходить через послідовність кроків ініціалізації - наприклад, налаштовує спостереження за даними, компілює шаблон, монтує екземпляр в DOM, оновлює DOM при зміні даних [11]. Між цими кроками викликаються функції, звані хукамі життєвого циклу, за допомогою яких можна виконувати свій код на певних етапах:

— Created - можливо використовувати після ініціалізації екземпляру веб додатку;

— Mounted - життєвий цикл звертання до котрого можливе після ініціалізації ViewModel та заміна моделі на модель інтерфейсу додатку;

— BeforeUpdate - подія яка відстежується під час оновлення екземпляру додатку;

— AfterUpdate - подія яка виконує функціонал додатку після оновлення екземпляру;

— BeforeDestroy - цикл який виконує функцонал до знищення екземпляру;

— AfterDestroy - життєвий цикл виконуючий функцію знищення екземпляру веб додатку.

Для зв'язування DOM з даними примірника Vue використовує синтаксис, заснований на HTML. Всі шаблони Vue є дійсним HTML-кодом, який можуть розпарсити все HTML-парсери і браузери.

Для роботи Vue компілює шаблони в render-функції віртуального DOM. У поєднанні з системою реактивності, Vue вміє визначати мінімальне число компонентів для повторної відтворення і застосовує мінімальну кількість маніпуляцій до DOM при зміні стану програми.

Директиви - це спеціальні атрибути з префіксом v-. Як значення вони приймають один вислів JavaScript (за винятком v-for, яку ми вивчимо далі). Директива реактивно застосовує до DOM зміни при оновленні значення цього виразу:

— V-bind - динамічно зв'язується з одним або декількома атрибутами;

— V-cloak - ховає «вусаті» вираження, поки не підтягнулися дані;

— v-if - умова для рендера елемента;

— V-else - позначає «else блок» для v-if;

— V-for - циклічно проходить масив об'єктів;

— V-model - пов'язує стан з input елементом;

— V-on - пов'язує слухача події з елементом;

— V-once - рендерить елемент тільки спочатку і більше не стежить за ним;

— V-pre - НЕ компілює елемент і його дочірні елементи;

— V-show - перемикає видимість елемента, змінюючи властивість CSS display.

Найбільш простий спосіб зв'язування даних - це текстова інтерполяція з використанням синтаксису Mustache (подвійних фігурних дужок):

```
<map-marker
v-for="(item, index) in state.markers"
:key="index"
:coordinate="item.coordinates"
:title="item.name"
>
<view :style="styles.marker">
<text>
{{ item.cost }}
</text>
</view>
</map-marker>
```

Вираз у фігурних дужках буде замінено значенням властивості state.markers відповідного об'єкта даних. Крім того, воно буде оновлено при будь-якій зміні цієї властивості. Даний приклад описає поняття реактивності мобільного додатку а додає динамічний функціонал та роботу.

4.1 Архітектура та класифікація Api.

За основу розробки архітектури була взята платформа REST Api.Це стиль архітектури програмного забезпечення для розподілених систем, таких як World Wide Web, який, як правило, використовується для побудови веб-служб. Термін REST був введений у 2000 році Роєм Філдінгом, одним з

авторів HTTP-протоколу. Системи, що підтримують REST, називаються RESTful-системами.

Як відбувається управління інформацією сервісу - це цілком і повністю ґрунтується на протоколі передачі даних. Найбільш поширений протокол звичайно ж HTTP. Таким чином, дії CRUD (Create-Read-Updtae-Delete) можуть виконуватися як з усіма 4-ма методами, так і тільки за допомогою GET і POST.

Основними методами запиту до серверу є:

— GET - для отримання (читання);

— POST - для створення;

— PUT - для зміни;

— DELETE - для видалення.

При розробці додатку була обрана бібліотека Axios для реалізації роботи Api.

Широко відома JavaScript-бібліотека. Вона являє собою HTTP-клієнт, заснований на Проміс і призначений для браузерів і для Node.js.

В додатку реалізовані наступні HTTP запити представлені на Таблиці 4.1

Таблиця 4.1 – Методи інтерфейсу Auth

Запит	Метод	Аргументи	Опис
/register	POST	email password	Створюється користувач в системі
/confirm email	POST	credentials	Етап підтвердження
/authenticate	POST	credentials	Аутентифікувати користувача

Запит	Метод	Аргументи	Опис
/login	POST	email, password	Аутентифікація користувача до облікового запису
/restore_ password	GET	email	Можливість користувача змінити пароль облікового запису
/token	GET		Токен облікового запису

Таблиця 4.2 – Методи інтерфейсу Events

Запит	Метод	Аргументи	Опис
/getLocation	GET		Місцезнаходження користувача
/getEvents	GET	credentials	Надає користувачу список усіх подій
/changeSearch _ Engine	PATCH	credential, location	Змінює критерії пошуку подій
/getOne	GET	event_id	Інформація події
/getComments	GET	event_id	Список коментарів
/setComment	POST	event_id, message	Залишае коментар користувача
/removeComm ent	DELETE	event_id	Видаляє коментар
/setLike	POST	event_id	Оцінка події

Запит	Метод	Аргументи	Опис
/restore_password	GET	email	Можливість користувача змінити пароль облікового запису

Реалізація виконання запитів додатку з використанням axios представлена на рисунку 4.1

```
export default {
  async getEvents(params) {
    let query = Object.entries(params).map(([key, val]) => `${key}=${val}`).join('&')
    return await axios.get(`/api/events/day-by-day-view?${query}`)
  },

  async getComments(params) {
    let query = Object.entries(params).map(([key, val]) => `${key}=${val}`).join('&')
    return await axios.get(`/api/events/get-comments?${query}`)
  },
```

Рисунок 4.1 — Створення запитів додатку

Кожен з представлених запитів визивається зі сховища додатку задля збереження даних. Методи представляють собою створення GET запросу на потрібний ендпоинт. Кожен метод приймає вхідні параметри які конвертуються у строку запиту задля формування критеріїв пошуку у базі даних.

4.2 Підключення інструменту Facebook SDK

Також при розробці інтерфейсу необхідно підключити інструменти використання Facebook. Для встановлення пакетів інструменту була використана наступна команда пакетного менеджеру - npm install vue-facebook.

Після завантаження пакетів необхідною дією було встановлення данного інструменту як глобального для використання в усіх необхідних компонентах інтерфейсу, реалізацію прелставолено на рисунку 4.2

```
import VueFacebook from 'vue-facebook';

Vue.use(VueFacebook)
```

Рисунок 4.2 — Підключення інструменту Facebook SDK

Налагодження та використання інструменту потребує створення загального сервісу обробки події та створення Api зв'язку з веб-сервером додатку та сервером Facebook. Основний запит на зв'язування користувача з платформою наведено на рисунку 4.3

```
this.FB.api('/me', 'GET', { fields: 'id,name,email,picture' },
  user => {
    this.personalID = user.id;
    this.email = user.email;
    this.name = user.name;
    this.picture = user.picture.data.url;
  }
)
```

Рисунок 4.3 — Запит синхронізації користувача з Facebook

Відображення аутентифікації користувача у соціальну мережу не відображається на інтерфейсі додатку але все одно цей функціонал реалізується. Facebook надає локальну форму авторизації користувача з встановленого інструменту, приклад форми користувача представлено на рисунку 4.4

Отже компоненти - це поділ коду, його инкапсулирование, на незалежні частини дозволяють повторно використовувати код. Компоненти утворюють деревоподібну ієрархію.

```
<facebook-login class="button"
        appId="420905468863679"
        @login="onLogin"
        @logout="onLogout"
        @get-initial-status="getUserData"
        @sdk-loaded="sdkLoaded">
    </facebook-login>
```

Рисунок 4.4 — Компонент авторизації користувача

4.3 Взаємодія компонентів інтерфейсу

Цілісність програмного коду на якісь окремі компоненти, які потім взаємодіють між собою, гарне рішення при побудові якісних додатків.Такі додатки легше підтримувати, адже код кожного компонента написаний так, щоб було якомога менше залежностей між компонентами. Адже коли перший код залежить від другого, і в цьому другому, щось змінилося, то можна очікувати, що в першому коді щось піде не так як очікувалося.

Vue.js дозволяє розділяти весь код програми на компоненти і навіть асинхронно завантажувати необхідний компонент в потрібний момент. Кожен компонент додатку має реалізовану структуру роботи директив, приклад представлено на рисунку 3.5.

```
<nb-container>
  <nb-header>
    <nb-left>
      <nb-button transparent :onPress="() => this.props.navigation.navigate('MainMap')">
        <nb-icon name="arrow-back" />
      </nb-button>
    </nb-left>
    <nb-body>
      <nb-title>Item</nb-title>
    </nb-body>
    <nb-right>
      <nb-button transparent :onPress="() => this.props.navigation.openDrawer()">
        <nb-icon name="arrow-back" />
      </nb-button>
    </nb-right>
  </nb-header>
```

Рисунок 4.5 — Приклад реалізації директив

Також на прикладі реалізована взаємодія компонентів та їх навігація.

На кнопку nb-button накладено обробник події onPress, це функція яка виконує навігацію між компонентами додатку.

Була впроваджена бібліотека готових компонентів для спрощення розробки додатку.

Vuesax - це рамка компонентів інтерфейсу, створених разом з Vuejs, щоб робити проекти легко і з неповторним і приємним стилем, vuesax створюється з нуля і розроблений для всіх типів розробників, починаючи від

любителя фронту бекенд, який хоче легко створити свій візуальний підхід до кінцевого споживача

Була розроблена структурна схема взаємодії та роботи компонентів додатку. Приклад структури наведений на рисунку 4.6

Рисунок 4.6 — Структурна схема взаємодії компонентів

Основною метою додатку - це простота використання та комфорт користувача. Для реалізації цього функционалу була використана бібліотека Sencha Ext JS

Бібліотека заснована на MVC JavaScript-інфраструктура. Цей інструмент розробки мобільних додатків забезпечує високий рівень відгуку програми. Це допомагає поліпшити задоволеність клієнтів.

Переваги бібліотеки в розробці додатку:

— написання в форматі HTML5 за допомогою набору для розробки (SDK) Sencha touch;

— код можна перевести на іншу мову програмування за допомогою іншого інструменту, такого як PhoneGap;

— інструмент корисний для розробки власних додатків без времязатрат на настройку;

— підтримується в браузерах на базі WebKit, включаючи популярні платформи Apple iOS і Google Android;

— раціоналізувати система конфігурації;

4.4 Управління станом сховища

Vuex - патерн і бібліотека управління станом для додатків на Vue.js. Він надає централізоване загальний стан для всіх компонентів в додатку і правила, що забезпечують передбачуване зміна стану. Це є основною метою обробки та зберігання даних з Api

Різновиди стану сховища:

— стан (State), яке є єдиним джерелом даних для компонентів.

— vue-компоненти (Vue-components), які по суті є лише декларативним відображенням стану;

— дії (Actions), які фільтрують подія, яка сталася, збирають дані з зовнішніх API і запускають потрібні мутації;

— мутації (Mutations) - єдина частина, яка може змінювати стан і, отримавши дані від Actions, застосовує їх на стані;

Приклад архітектури сховища представлено на Рисунок 4.7

Навіщо потрібна правильна архітектура сховища? Сховище задає структуру роботи усього додатку. Створення запитів для отримання даних, відображення компонентів та переходів між ними. Структура розробленого сховища представляє собою модульне створення екземпляру сховища.

Основі переваги використання сховища:

— сховище Vuex - реактивне. Як тільки компоненти отримують з нього стан, вони будуть реактивно оновлювати свої уявлення кожен раз, коли стан змінюється.

— компоненти не можуть безпосередньо змінювати стан сховища. Єдиний спосіб змінити стан сховища - явно зафіксувати мутації. Це гарантує, що кожна зміна стану залишає відстежуємо запис, що полегшує налагодження та тестування програми.

— сховище Vuex дає вам загальну картину стану, як все пов'язано і впливає на додаток.

— простіше підтримувати і синхронізувати стан між кількома компонентами, навіть якщо ієрархія компонентів змінюється.

— vuex уможливлює безпосередню взаємодію компонентів один з одним.

— якщо компонент знищений, стан в сховище Vuex залишиться недоторканим.

```
import Vue from 'vue'
import Vuex from 'vuex'

import state from "./state"
import getters from "./getters"
import mutations from "./mutations"
import actions from "./actions"

import project from './project'
import rankTraking from './rankTraking'

Vue.use(Vuex)

// import moduleTodo from './todo/moduleTodo.js'
// import moduleCalendar from './calendar/moduleCalendar.js'
// import moduleChat from './chat/moduleChat.js'
// import moduleEmail from './email/moduleEmail.js'
import moduleAuth from './auth/moduleAuth.js'
import moduleECommerce from './eCommerce/moduleECommerce.js'


export default new Vuex.Store({
    getters,
    mutations,
    state,
    actions,
    modules: {
        // todo: moduleTodo,
        // calendar: moduleCalendar,
        // chat: moduleChat,
        // email: moduleEmail,
        auth: moduleAuth,
        // eCommerce: moduleECommerce,
        project: project,
        // rankTraking: rankTraking
    },
    strict: process.env.NODE_ENV !== 'production'
})
```

Рисунок 4.7 — Архітектура впровадження модуля в сховище

4.5 Навігація інтерфейсу

Vue надає різні способи застосування анімаційних ефектів, коли елементи намальовані, оновлені або видалені з DOM. Вони включають в себе інструменти для:

— автоматичного застосування класів для CSS-переходів і анімацій інтеграції сторонніх бібліотек для CSS-анімації, таких як Animate.css;

— використання JavaScript для маніпуляції DOM-му;

— інтеграції сторонніх JavaScript бібліотек для анімацій, таких як Velocity.js;

Підключення нових компонентів до роботи навігаційної системи представлено на рисунку 4.8.

```
const Drawer = createDrawerNavigator(
  {
    Main: { screen: Main },
    MainMap: { screen: MainMap },
    EventId: { screen: EventId },
    Home: { screen: HomeScreen },
    Anatomy: { screen: AnatomyScreen },
    Actionsheet: { screen: ActionSheetScreen },
    Header: { screen: HeaderScreen },
    Footer: { screen: FooterScreen },
    NHBadge: { screen: BadgeScreen },
    NHButton: { screen: ButtonScreen },
    NHCard: { screen: CardScreen },
    NHCheckbox: { screen: CheckboxScreen },
    NHDeckSwiper: { screen: DeckSwiperScreen },
    NHFab: { screen: FabScreen },
    NHForm: { screen: FormScreen },
    NHIcon: { screen: IconsScreen },
    NHLayout: { screen: LayoutScreen },
    NHList: { screen: ListScreen },
    // ListSwipe: { screen: ListSwipeScreen },
    NHRadio: { screen: RadioScreen },
    NHSearchbar: { screen: SearchScreen },
    NHPicker: { screen: PickerScreen },
    Segment: { screen: SegmentScreen },
    NHSpinner: { screen: SpinnerScreen },
    NHTab: { screen: TabScreen },
    NHThumbnail: { screen: ThumbnailScreen },
    NHToast: { screen: ToastScreen },
```

Рисунок 4.8 — Підключення компонентів до додатку

Для збільшення функціоналу анімації та переходів було використано бібліотеку Tailwind.js.

Tailwind CSS - це спеціально налаштована система низького рівня CSS, яка надає всі необхідні вам будівельні блоки для створення замовлених конструкцій без будь-яких дратівливих самовпевнених стилів, з якими вам доведеться боротися, щоб перекрити.

Tailwind HTML - надає весь спектр готових налаштованих шаблонів готового HTML конструктора.

Висновки до розділу

В даному розділі був створеній інтерфейс додатку Find&Go. При розробці інтерфейсу була використана бібліотека NativeScript, яка надає можливість розробляти мобільний додаток на основі веб-додатку з використанням JavaScript, HTML, CSS, також з використанням цієї бібліотеки була реалізована можливість збирати додаток у мобільну версію. Була обрана бібліотека-обгортка Vue.js - надає можливість взаємодії усіх компонентів та станів додатку.

Була створена діаграма взаємодії компонентів інтерфейсу та взаємодія даними між ними, також додана обротка переходів між компонентами та загальний стан хвища проекту який керує даними з запитів та зберігає їх у собі. Були підключені бібліотеки з готовими шаблонами HTML та CSS конструкції, що надає комфортне будування архітектури взаємодії між компонента та їх видом.

5 ІНСТРУКЦІЯ КОРИСТУВАЧА

Даний розділ представляє точний опис дій користувача на кожному екрані.

Для того щоб розпочати роботу з додатком, спочатку користувач повинен завантажити мобільний додаток до свого смартфону. Опублікований додаток у AppStore зображено на рисунку 5.1.

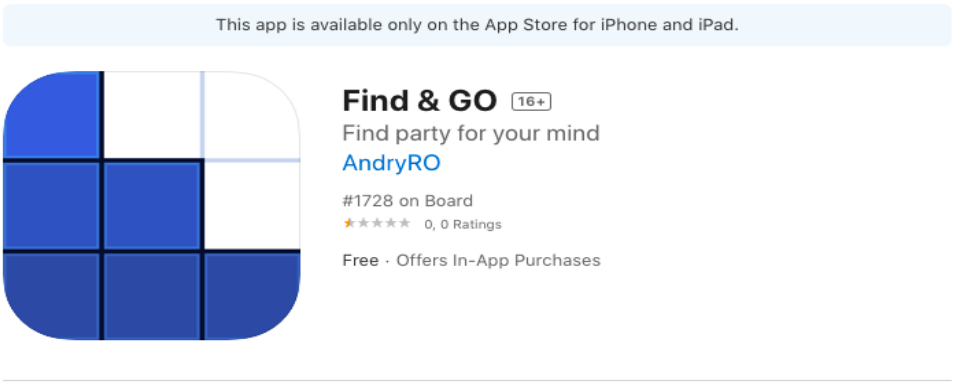

Рисунок 5.1 — Опублікований додаток у AppStore

Користувач має можливість оцінити додаток та підняти його у рейтингу додатків. Знайти схожі за посилом додатки.

Одразу після завантаження додатку у користувача є можливість зареєструватися декількома способами: використовуючи пошту або Facebook Account.

Якщо реєстрація проходить через пошту, форма перевіряє кожну з валідних комірок. Мобільний телефон повинен відповідати телефонному коду держави та бути дійсним, Email має бути валідним та мати обов’язковий символ @, пароль повинен бути не менше 5 символів та мати велику літеру. Після відправки форми на номер телефону користувача буде відправлене СМС повідомлення з підтвердження реєстрації.

Після входу до облікового запису користувачу надається повний функціонал додатку. Стартова сторінка додатку має кнопку запуску роботи функціоналу додатку, як зображено на рисунку 5.3.

Рисунок 5.3 — Стартова сторінка додатку

Наступний екран додатку залежить від статусу користувача, якщо запуск додатку виконується вперше то ініціалізується екран форми заповнення побажань користувача. Форма, що зображена на рисунку 5.4, представляє собою 5 загальних комірок які обов’язкові для заповнення. Локація, користувач може обрати точну локацію пошуку подій або за місцем знаходження. Віковий та грошовий діапазон подій, основна валюта пошуку підбирається в залежності від коду номера телефону користувача. Основною коміркою є - тип музики, вибір користувачу надається з вже з сформованих шаблонів пошуку.

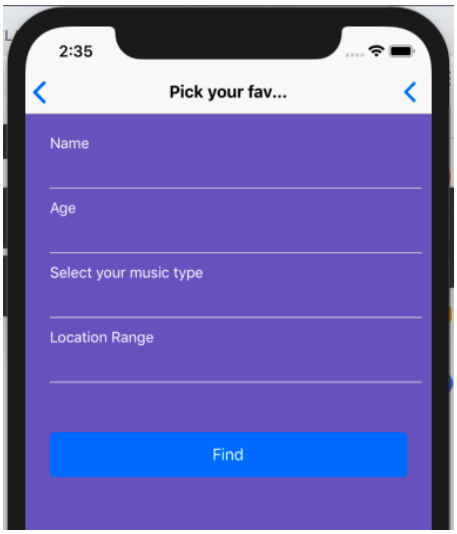

Рисунок 5.4 — Форма побажань користувача

Якщо користувач вже має створений шаблон пошуку, він має можливість обрати його або створити новий, кожен обліковий запис може створити до 5 шаблонів пошуку подій. Наступний екран, який зображенй на рисунку 5.5, відповідає за відображення на мапі подій та їх списку.

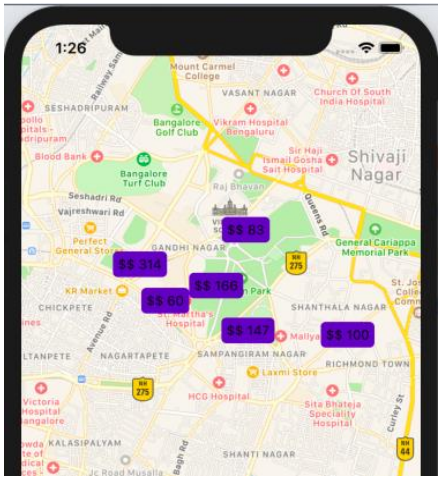

Рисунок 5.5 — Мапа з подіями

У наведеному рисунку представлено події за обраною темою пошуку, діапазон пошуку відображаються в залежності від розміру мапи. Користувач може натиснути на цінник події що відображають більш детальну інформацію події, а саме її назву та час. Також у списку подій зображеному на рисунку 5.6 обрана подія підсвітится.

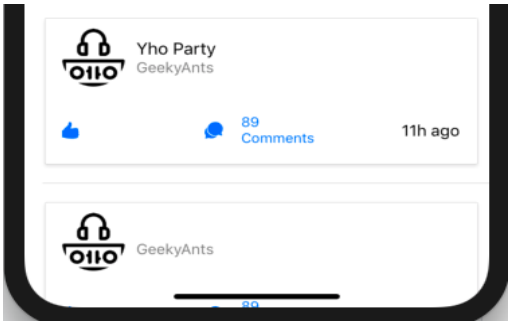

Рисунок 5.6 — Список подій

Кожна подія у списку підключена до соціальної мережі Facebook через котру користувач має можливість реагувати та залишати коментарі на подію. Також імплементована можливість додавання події у розділ бажаної. Для того щоб подивитися повну інформацію обраної події користувач повинен клацнути на неї. Відкриється модальне вікно.

Екран відображає повну інформацію події та коментарі інших користувачів. Реалізована кнопка придбання квитка, якщо подія передбачає наявність квитка. Придбання можливе при підключеній картці на смартфоні, як показано на рисунку 5.7.

Рисунок 5.7 — Придбання ApplePay

Після придбання квитків на поточну подію, реквізити та сам квито буде відправлений на пошту користувача.

Висновки до розділу

У даному розділі була описана пошагові етапи взаємодії користувача та додатку. Розглянутий представлений функціонал, інтерфейс додатку має дуже чіткі сторінки та їх загальний сенс. Простий та зрозумілий інтерфейс водночас багатофункціональні компоненти.

ВИСНОВКИ

Метою у ході розробки дипломного проекту були розглянуті основні принципи створення мобільних додатків, а також огляд існуючих програмних рішень.

Проведено аналіз та виконана постановка основних задач дипломного проектування а саме:

- розроблений додаток Find&Go;
- реалізована авторизація користувача;
- створений та налаштований веб-сервер;
- підключення та налаштування бази даних з моделями серверу;
- підключення платіжної системи
- інтеграція мапи
- підключення соціальної мережі

Створений мобільний додаток Find&Go, яки надає необхідну інформацію користувачу за поточною подією також були використанні необхідних інтсрументи платформи IOS для розробки додатку.

Налаштована система безпеки авторизації користувача з використання зовнішніх бібліотек а саме: JWT токен, Node SS. Кожна з бібліотек виконує свій необхідний функціонал за для беспечного використання та зберігання налаштувань та даних користувача.

Розробленийй веб-сервер написаний з використання платформи Node.js, була нашаштована взаэмодія запитів до інтерфейсу додатку з використанням зовнішніх бібліотк а саме: express.js та методології rest api. Також до веб-серверу були інтегровані зовнішній необхідний функціонал: Portmone та Facebook SDK, які реалізують створення платіжної системи та використання соціальної мережі як частини мобільного додатку .

Була створеня та налаштована база даних для зберігання та використовування даних записних або використаних користувачем. За основу платофрми бази даних була використана база MySQL. Основним посилом створення бази даних було правильне налаштування взаємодії усіх таблиц та спілкування з ними, за для цього була використана методологія спілкування з базою як з звичайним об’єкто классу. Це впровадження

спрощує налаштування та спілкування з базою даних через моделі веб-серверу.

Налаштована та підключена платіжна система дає можливість придбання квитків до цікавлющої події користувача. Реалізація платіжнох системи поділяється на два етапи: налаштування на стороні веб-серверу та на стороні інтерфейсу. Основним принципом роботи плітжнох системи є використання окремого модуля який надає платіжна система та налагодження його налагодження.

Важливим питання роботи додатку було створення мапи та відображення міток подій для користувача. За основу мапи була взята зовнішня бібліотека Google Maps, яка не має на цей час конкурентів з аналогічним функціоналом. Інтеграція мапи було реалізована на стороні інтерфейсу та були використані усі необхідні інструменти для коретного відображення подій.

Реалізовано підключення та налаштування соціальної мережі для можливості спілкування користувачів додатку. Був використаний пакет інструментів Facebook SDK. Основний функціонал якого полягає в можливості коментування поточної події та додавання події до бажаних.

Отже, платформа створена з урахуванням розглянутих недоліків існуючих проектів, а саме:

- можливість придбання квитків;
- адаптивне відображення подій в залежності від місцезнаходження;
- інтеграція соціальної мережі;
- можливість створення своїх подій.

Розроблений додаток має практичне застосування і вже розробляється в більше інтенсивному темпі, проходить пошук інвесторів. Додаток у ближайший час потрапить на усі платформи для завантаження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Исследование актуальных программ для разработки дизайна мобильных приложений [Електронний ресурс] Режим доступу до ресурсу: https://moluch.ru/archive/286/64523/.
2. Стаття актуальності мобільних додатків [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.scienceeducation.ru/ru/article/view?id=15957/
3. Социальные мобильные приложение [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://woxapp.com/ru/industries/social-app-development/.
4. NativeScript - нативний фреймворк мови JavaScript [Електронний ресурс]- Доступ: https://docs.nativescript.org/
5. Vue.js - фреймворк для розробки інтерфейсу [Електронний ресурс]- Доступ: https://vuejs.org/v2/guide/
6. Portmone - платіжна система [Електронний ресурс] - Доступ: https://docs.portmone.com.ua/docs/uk/PortmoneHostToHostUkr/
7. Node.js – фреймворк розробки веб-серверу [Електронний ресурс]
8. Xcode – середа розробки мобільного додатку [Електроний ресурс] - Доступ: https://developer.apple.com/documentation/xcode/
9. IOS. Приемы програмирования – Питер: Нахавандипур В, 2014. – 432 с.
10. Vue.js в действии – Питер: Хэнчетт Э, 2019. – 304 с. – (Питер). – (978-5-4461-1098-8; кн. 978).
11. Рабаи, Ж. М. Методология проектирования, 2-е издание / Жан Морис Рабаи,., 2016. – 916 с. – (2). – (978-5-8459-1116-2; кн. 978).
12. Марейн Х. Выразительный JavaScript / Хавербеке Марейн. – Питер: Питер, 2014. – 480 с. – (3).
13. IOS Simulator – емулатор мобільного додатку середовища Xcode [Електронний ресурс] – Доступ:

https://developer.apple.com/library/archive/documentation/IDEs/Conceptual/iOS_Simulator_Guide/GettingStartedwithiOSSimulator/GettingStartedwithiOSSimulator.html

14. Архітектура платіжних систем [Електроний ресурс] – Доступ: https://habr.com/ru/company/oleg-bunin/blog/354824/
15. SQL: полное руководство – Лондон: Диалектика-Вильямс, 2019. – 923 с. – (1). – (1; кн. 2147483647).
16. Дудзяний І. М. Об'єктно-орієнтоване моделювання програмних систем / І. М. Дудзяний. – Львів: Видавничий центр ЛНУ імені Івана Франка, 2007. – 108 с.
17. What exactly is Node.js? [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/.
18. Sandro P. Mastering Node.js / Pasquali Sandro. – Лівері: Packt Publiching Ltd, 2013. – 329 с.

Додаток А

```
<template>
  <form-wizard
    color="rgba(var(--vs-primary), 1)"
    errorColor="rgba(var(--vs-danger), 1)"
    :title="null"
    :subtitle="null"
    finishButtonText="Save"
  >
    <tab-content title="Step 1" class="mb-2" :before-change="submitStep1">
      <step-1 v-on:onChange="onChange" :domain="domain"
:project_name="project_name" />
    </tab-content>
    <tab-content title="Step 2" class="mb-2" :before-change="submitStep2">
      <step-2
        :options="options"
        :domain="domain"
        :language_id="language_id"
        :track_type="track_type"
        :keywords="keywords"
        :is_default="is_default"
        v-on:onInputKeywords="onInputKeywords"
        :project_search_engines="project_search_engines"
        v-on:addProjectSearchEngine="addProjectSearchEngine"
        v-on:removeProjectSearchEngine="removeProjectSearchEngine"
        v-on:toggleDefaultProjectSearchEngine="toggleDefaultProjectSearchEngine"
        v-on:selectTrackType="selectTrackType"
        v-on:selectLanguage="selectLanguage"
      />
```

```
    </tab-content>
    <tab-content title="Step 3" class="mb-2">
      <step-3 />
    </tab-content>
  </form-wizard>
</template>
<script>
import axios from "../../../http/axios";
import { Validator } from "vee-validate";
import { mapActions, mapGetters } from "vuex"
import { FormWizard, TabContent } from "vue-form-wizard";
import "vue-form-wizard/dist/vue-form-wizard.min.css";
    async submitStep1() {
      return new Promise((resolve, reject) => {
        this.$validator.validateAll("step-1").then(async result => {
          if (result && !this.isStep1Submitted) {
            const { domain, project_name, project_id } = this
            const params = {
              domain,
              project_name,
              project_id
            }
            if (this.project_id) {
              await this.updateProject(params)
            } else {
              await this.addProject(params).then((res) => {
                this.project = res
              })
            }
```

```
        this.isStep1Submitted = true
        resolve(true)
      } else {
        reject("correct all values");
      }
    })
  })
},
async searchEngine(data) {
  this.project_search_engines_new.map(async _ => {
    const language = this.options.languages.find(
      ({ text }) => text === _.language
    )
    const params = {
      project_id: this.project.id,
      engine: _.search_engine_domain,
      country: _.country,
      language_key: language.key
    }
    const data = await this.addSearchEngine(params)
    const trackTypeObj = this.options.track_types.find(
      ({ id }) => id === this.track_type
    );
    const fields = {
      project_id: this.project.id,
      track_type: trackTypeObj.text,
      selected_search_engine: data.id
    }
    await this.updateProject(fields)
```

```
  })
},
async keywordAdd() {
  if (this.keywords !== '') {
      const params = {
      project_id: this.project.id,
      keywords: this.keywords
    }
    await this.addKeywords(params)
    // todo keywords adding !!
  }
  return
},
submitStep2() {
  return new Promise((resolve, reject) => {
    this.$validator.validateAll("step-2").then(async result => {
      if (result) {
        if (!this.isStep2Submitted) {
          // submit all selected search engines
          this.searchEngine()
          this.keywordAdd()
        }
        resolve(true);
      } else {
        reject("correct all values");
      }
    });
  });
}
```

```
  }
};
</script>

<template>
        <div id="app" :class="vueAppClasses">
                <router-view @setAppClasses="setAppClasses" />
        </div>
</template>

<script>
import themeConfig from '@/../themeConfig.js'
import jwt        from "@/http/requests/auth/jwt/index.js"
export default {
  data() {
    return {
      vueAppClasses: [],
    }
  },
  watch: {
    '$store.state.theme'(val) {
      this.toggleClassInBody(val)
    },
    '$vs.rtl'(val) {
      document.documentElement.setAttribute("dir", val ? "rtl" : "ltr")
    }
  },
  methods: {
    toggleClassInBody(className) {
```

```
      if (className == 'dark') {
        if (document.body.className.match('theme-semi-dark'))
document.body.classList.remove('theme-semi-dark')
        document.body.classList.add('theme-dark')
      }
      else if (className == 'semi-dark') {
        if (document.body.className.match('theme-dark'))
document.body.classList.remove('theme-dark')
        document.body.classList.add('theme-semi-dark')
      }
      else {
        if (document.body.className.match('theme-dark'))
document.body.classList.remove('theme-dark')
        if (document.body.className.match('theme-semi-dark'))
document.body.classList.remove('theme-semi-dark')
      }
    },
    setAppClasses(classesStr) {
      this.vueAppClasses.push(classesStr)
    },
    handleWindowResize() {
      this.$store.commit('UPDATE_WINDOW_WIDTH', window.innerWidth)

      // Set --vh property
      document.documentElement.style.setProperty('--vh', `${window.innerHeight *
0.01}px`);
    },
    handleScroll() {
      this.$store.commit('UPDATE_WINDOW_SCROLL_Y', window.scrollY)
```

```
    }
  },
  mounted() {
    this.toggleClassInBody(themeConfig.theme)
    this.$store.commit('UPDATE_WINDOW_WIDTH', window.innerWidth)
    let vh = window.innerHeight * 0.01;
    // Then we set the value in the --vh custom property to the root of the document
    document.documentElement.style.setProperty('--vh', `${vh}px`);
  },
  async created() {

    // jwt
    jwt.init()
    let dir = this.$vs.rtl ? "rtl" : "ltr"
    document.documentElement.setAttribute("dir", dir)
    window.addEventListener('resize', this.handleWindowResize)
    window.addEventListener('scroll', this.handleScroll

    // Auth0
    try      { await this.$auth.renewTokens() }
    catch (e) { console.error(e) }
  },
  destroyed() {
    window.removeEventListener('resize', this.handleWindowResize)
    window.removeEventListener('scroll', this.handleScroll)
  },
}
</script>
import axios from '../../axios'
```

```
export default {
  getOne(id) {
    return axios.get(`/api/projects/${id}`)
  },
  add(data) {
    return axios.post('/api/projects', data)
  },
  update(data) {
    const { project_id } = data
    return axios.patch(`/api/projects/${project_id}`, data)
  },
  addSearchEngine(data) {
    const { project_id } = data
    return axios.post(`/api/project-search-engine?project_id=${project_id}`, data)
  },
  removeSearchEngine(data) {
    return axios.delete(`/api/project-search-engine/${data}`)
  },
  addKeywords(data) {
    const { project_id } = data
    return axios.post(`/api/project-keyword?project_id=${project_id}`, { keyword:
data.keywords })
  }
}
import axios from "../../../axios/index.js";
import store from "../../../../store/store.js";

// Token Refresh
let isAlreadyFetchingAccessToken = false;
```

```
let subscribers = [];

function onAccessTokenFetched(access_token) {
    subscribers = subscribers.filter(callback => callback(access_token));
}
function addSubscriber(callback) {
    subscribers.push(callback);
}
export default {
    init() {
        axios.interceptors.request.use(config => {
            const token = localStorage.getItem("access_token");
            if (token !== null) {
                config.headers.common["Authorization"] = "Bearer " + token;
            }
            return config;
        });
        axios.interceptors.response.use(
            response => response,
            async error => {
                const status = error.response ? error.response.status : null;
                if (status !== 401) {
                    return Promise.reject(error);
                }
                if (error.response.config.url === "/api/logout") {
                    return Promise.resolve();
                }
                if (!isAlreadyFetchingAccessToken) {
                    isAlreadyFetchingAccessToken = true;
```

```
                const token = await store.dispatch("auth/fetchAccessToken");
                isAlreadyFetchingAccessToken = false;
                onAccessTokenFetched(token);
            }
            const retryOriginalRequest = new Promise(resolve => {
                addSubscriber(access_token => {
                    error.response.config.headers.Authorization = `Bearer
${access_token}`;
                    resolve(axios(error.response.config));
                });
            });

            return retryOriginalRequest;
        }
    );
  },

  login(email, pwd) {
    return axios.post("/api/login", { username: email, password: pwd });
  },
  logout() {
    return axios.post("/api/logout");
  },
  register(name, email, pwd) {
    return axios.post("/api/register", { name, email, password: pwd });
  },
  refreshToken() {
    return axios.post("/api/token/refresh", {
```

```
            refresh_token: localStorage.getItem("refresh_token")
        });
    }
};

/*=============================================================
==============================
  File Name: main.js
  Description: main vue(js) file
  ----------------------------------------------------------------------------------
  Item Name: Vuesax Admin - VueJS Dashboard Admin Template
  Author: Pixinvent
  Author URL: hhttp://www.themeforest.net/user/pixinvent
=============================================================
==============================*/
import Vue from 'vue'
import App from './App.vue'



// Vuesax Component Framework
import Vuesax from 'vuesax'

Vue.use(Vuesax)

// axios
import axios from "./axios.js"
Vue.prototype.$axios = axios

// API Calls
```

```
import "./http/requests"

// mock
import "./fake-db/index.js"

// Theme Configurations
import '../themeConfig.js'

// Firebase
// import '@/firebase/firebaseConfig'

// Auth0 Plugin
import AuthPlugin from "./plugins/auth"
Vue.use(AuthPlugin);

// ACL
import acl from './acl/acl'

// Globally Registered Components
import './globalComponents.js'

// Vue Router
import router from './router'

// Vuex Store
import store from './store/store'

// i18n
import i18n from './i18n/i18n'
```

```
// Vuesax Admin Filters
import './filters/filters'

// Clipboard
import VueClipboard from 'vue-clipboard2'
Vue.use(VueClipboard);

// Tour
import VueTour from 'vue-tour'
Vue.use(VueTour)
require('vue-tour/dist/vue-tour.css'

// VeeValidate
import VeeValidate from 'vee-validate'
Vue.use(VeeValidate);

// Google Maps
import * as VueGoogleMaps from 'vue2-google-maps'
Vue.use(VueGoogleMaps, {
   load: {
      // Add your API key here
      key: 'AIzaSyB4DDathvvwuwlwnUu7F4Sow3oU22y5T1Y',
      libraries: 'places', // This is required if you use the Auto complete plug-in
   },
})

// Vuejs - Vue wrapper for hammerjs
import { VueHammer } from 'vue2-hammer'
```

```
Vue.use(VueHammer)

export default actions

/*==============================================================
==============================
  File Name: store.js
  Description: Vuex store
  ----------------------------------------------------------------------------
  Item Name: Vuexy - Vuejs, HTML & Laravel Admin Dashboard Template
  Author: Pixinvent
  Author URL: http://www.themeforest.net/user/pixinvent
==============================================================
==============================*/
import Vue from 'vue'
import Vuex from 'vuex'
import state from "./state"
import getters from "./getters"
import mutations from "./mutations"
import actions from "./actions"
import project from './project'

Vue.use(Vuex)

// import moduleTodo from './todo/moduleTodo.js'
// import moduleCalendar from './calendar/moduleCalendar.js'
// import moduleChat from './chat/moduleChat.js'
// import moduleEmail from './email/moduleEmail.js'
import moduleAuth from './auth/moduleAuth.js'
```

```
import moduleECommerce from './eCommerce/moduleECommerce.js'

export default new Vuex.Store({
    getters,
    mutations,
    state,
    actions,
    modules: {
        auth: moduleAuth,
        eCommerce: moduleECommerce,
        project: project,
    },
    strict: process.env.NODE_ENV !== 'production'
})
```